NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

CONCEPT-FLOW DIAGRAMS: METHOD FOR DESIGN OF COMPUTER-AIDED INSTRUCTION

by

DAWN MARIE MASKELL

MARCH 1992

Thesis Advisor: TIMOTHY J. SHIMEALL

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: SPAWARS; 8b. OFFICE SYMBOL (if applicable): PMW-183	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code): Washington D.C. 20363-5100	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
CONCEPT-FLOW DIAGRAMS: METHOD FOR DESIGN OF COMPUTER-AIDED INSTRUCTION

12. PERSONAL AUTHOR(S)
MASKELL, DAWN M.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM TO	MARCH 1992	150

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Concept-Flow Diagram, Mastery Test

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Current software design techniques are organized around either data transformation or stimulus-response control flow abstractions. Neither of these approaches apply to the flow of presentation and answer analysis that characterizes computer-aided instruction. This thesis introduces a new design abstraction, concept-flow, and technique that exploits it in the design of tutorial software. The design technique uses concept-flow diagrams, which highlight presentation of information and verification of user comprehension.The technique is explained through application to a tutorial on the physics of underwater sound. The design and implementation of a prototype concept-flow interpreter are presented. This design technique and the associated interpreter allow for rapid construction of highly flexible computer-based tutorial strategies, useful for both traditional CAI applications and for more efficient help-sequence design in interactive systems.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
[X] U.CLASSIFIED/UNLIMITED [ ] SAME AS RPT [ ] DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: SHIMEALL, TIMOTHY J.	22b. TELEPHONE (Include Area Code): (408) 646-2509	22c. OFFICE SYMBOL: CS/Sm

Approved for public release; distribution is unlimited

CONCEPT-FLOW DIAGRAMS: METHOD FOR DESIGN OF COMPUTER-AIDED INSTRUCTION

by

Dawn Marie Maskell
Lieutenant, United States Navy
B. A., University of California, Los Angeles, 1985

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1992

Author: Dawn Marie Maskell
Dawn Marie Maskell

Approved By: Timothy J. Shimeall
Timothy J. Shimeall , Thesis Advisor

Alan B. Coppens
Alan B. Coppens, Second Reader

Robert B. McGhee
Robert B. McGhee, Chairman,
Department of Computer Science

ABSTRACT

Current software design techniques are organized around either data transformation or stimulus-response control flow abstractions. Neither of these approaches apply to the flow of presentation and answer analysis that characterizes computer-aided instruction. This thesis introduces a new design abstraction, concept-flow, and technique that exploits it in the design of tutorial software. The design technique uses concept-flow diagrams, which highlight presentation of information and verification of user comprehension.The technique is explained through application to a tutorial on the physics of underwater sound. The design and implementation of a prototype concept-flow interpreter are presented. This design technique and the associated interpreter allow for rapid construction of highly flexible computer-based tutorial strategies, useful for both traditional CAI applications and for more efficient help-sequence design in interactive systems.

TABLE OF CONTENTS

I. INTRODUCTION

A. THE PROBLEM

More and more, instructors are using computers as instructional tools both in and out of the classroom environment. The use of a computer as an instructional tool is referred to as Computer-Aided Instruction (CAI), "a process in which the computer is actually the prime deliverer of the instruction . . . ." (Burke, 1982, p. 16)

> The educational promise of CAI lies in its ability to individualize and personalize the instructional process. CAI lessons can test as well as tutor by encouraging students to become active participants in their own learning. Students work at their own pace while the computer monitors their progress. Students are kept informed of their progress through intermediate response and achievement summaries. (Marks, 1981, p. 228)

Current techniques for design of instructional software tend to produce computer-driven tutorials. The computer system drives the presentation of a specific topic and determines to what degree the presentation of that concept will occur even if the user already has a grasp of the concept. The computer presentation of the lesson is an almost linear process; presentation of the concepts is in a specific order with only limited or no deviation to this order allowed. Thus, current techniques do not simulate the real-world teaching environment. In a real-world teaching environment, the teacher determines the level of student comprehension; the teacher determines when the student has difficulty understanding the concept and can therefore digress to a level of comprehension upon which to build and teach the concept; and the teacher determines when the student understands a concept, when not to continue dwelling on the concept at hand and when to move on to a new concept presentation.

Current software design techniques do not allow for the proper representation of the concept-flow needed to simulate the real-world teaching environment in CAI. They aid in determining the control flow, when to execute a process, and the data flow, how data is passed between processes, within the system. Both of these techniques are useful in the software engineering field but are not very helpful with CAI design. There is a need for a design technique that allows the user to drive the concept presentation. Therefore, we want to design a technique that allows for a closer simulation of the teacher-student interaction environment.

B. THE NEEDS

To simulate the real-world teacher-student environment, the computer system must determine and detect when the lesson plan needs adjustment to better suit the needs of the user. The program must make the determination, as does a teacher, based upon user response. There are two ways of making this determination: either the user explicitly claims no knowledge or the user specifically claims knowledge that requires verification to ensure expertise.

If the user claims little or no knowledge of a concept, then presentation of the tutorial should automatically occur. Once the user initiates the program and begins demonstrating an understanding of concepts through task accomplishment, an actual explanation of a concept should occur only when the user demonstrates or claims unfamiliarity with that concept. When the user has trouble comprehending a concept, the program should backtrack until a level of comprehension is found. Basing the flow of the lesson upon this level of comprehension, the program then begins from this concept and moves forward. As Krendl suggests, this allows the novice user to benefit from structure, systematic presentation of material, and opportunity for practice. (Krendl, 1988, p. 371)

"High aptitude students are more likely to benefit when they can control the pace, amount of practice, level of difficulty, and style of instruction to suit their own needs." (Krendl, 1988, p. 371) Therefore, if the user shows or claims knowledge of a concept, omission of a tutorial of the concept should occur allowing the user to progress onto the next concept. This will prevent the user from experiencing the boredom produced by forcing him/her to cycle through a tutorial he/she already understands. An expert user should be able to drive the presentation forward from concept to concept after demonstrating an understanding of each concept through successful task accomplishment. Therefore, the user, instead of the computer, is guiding the concept presentation.

Software designers need to allow for this varying concept presentation flow in the program design. The program should allow a user with an arbitrary level of comprehension to traverse through the program dealing only with the presentation of concepts when they are not understood. Hence, in order to allow for this tutorial presentation, the concept presentation must encourage minimal ordering constraints.

Alfred Bork, a leader in the pedagogical development of computer-based learning, uses pedagogical flowcharts to diagram the presentation-flow of a tutorial (electronic mail from Bork). These flowcharts are non-hierarchical, informal, and are difficult to translate into software. There is a need for a formal diagram that more explicitly displays the hierarchy of the presentation and that leads to a natural translation into software.

C. A SOLUTION: CONCEPT-FLOW DIAGRAM

To support a minimal-ordering design, we introduce a new diagram to the structured analysis design methodology. The new tool is called a Concept-Flow Diagram (CFD). A CFD is for use to aid in the design of CAI by software engineers, computer scientists and anyone involved in the process of designing a tutorial of any type. The CFD is a high-level diagram and is for use in conjunction with pedagogical design, a series of instructional goals, and Dataflow Diagrams (DFD), the connections between program subunits.

1. Brief Overview of the Design Methodology

Before attempting to design the CFD, the designers must first attempt a pedagogical design. It is during this stage that the instructional materials are fully specified from an instructional point of view (Bork, p. 106). From the pedagogical design, specific concepts that need presentation should be apparent.

> A general, but less will-defined, method is to use the educational objectives and task analyses to subdivide the course into a set of concepts and techniques which have to be learned. These can be partitioned into a series of levels, depending on their complexity, and usually checks are made to ensure that the student has reached a satisfactory standard of mastery before he is allowed to continue to higher levels. (Walker, 1984, p. 45)

These concepts translate into a module or bubble in the CFD. The CFD represents the forward and the backward presentation of these concept modules. Hartley argues for ". . . a more comprehensive representation of the student's knowledge state on which to base decision making . . . ." (Walker, 1984, p. 39) Therefore, at the start of a new concept module, the user is given a task or comprehension test to complete via a mastery test (MT). This allows for an active involvement in learning, which is necessary for effective learning and achieving desired outcomes (Levin, 1981, p. 1). Performance upon completion of this task determines the user's level of comprehension. If the user demonstrates comprehension or satisfactorily completes the task, the system moves on to the next task or instruction-related exercise. If the user does not complete the task or cannot pass the comprehension test, the feedback should ". . . locate errors and provide information so that the learner can put them right. . . ." (Walker, 1984, p. 43) and also provide ". . . corrective procedures by which gaps in learning, mistakes, and misunderstandings can be relearned or corrected." (Levin, 1981, p. 16) The system then reverts to a tutorial and presentation of the concept. The idea is to introduce the tutorial only when needed. Upon successful completion of the MT for the specific concept module, flow moves on to the next concept module. This is in keeping with Bork's suggestion that pretesting and post-testing be included in the tutorial to make individualization possible. (Bork, p. 77)

2. Symbols Used

We based the symbols used in the CFD on Dataflow Diagrams to allow for easy understanding and translation of the new diagram. Refer to Yourdon's recent work (Yourdon, 1989, pp. 139-187) for a detailed explanation of the DFD.

a. Rectangle

A rectangle represents external entities, or terminators, with which the system communicates. A terminator is usually a person, a group of people, or another system outside the control of the system modeled. The systems analyst cannot change the contents, organization or internal procedures associated with the terminators. (Yourdon, 1989, pp. 155-156, pp. 345-347) This is the same as a terminator in a DFD.

b. Circle

A circle represents concept that is to be presented. (Also referred to as concept bubble). Circles are decomposed into further Concept-Flow Diagrams as needed to detail the concept-flow forming a design hierarchy.

c. Arrow

An arrow represents concept-flow and direction. It points to the next concept for introduction and/or MT to be given. An arrow indicates the dependency of concepts or modules from one part of the lesson to another.

d. Octagon

An octagon represents a series of tasks that form a MT. Octagons may be decomposed into further Concept-Flow Diagrams to show task sequencing and help presentation.

3. Labelling of Symbols Used

The name given to each concept should be specific enough to give the user of the diagram an idea of exactly what type of information is presented in the concept presentation. The name of the MT should be the exact same as the concept or concepts that the MT is testing for comprehension. Arrows do not get labelled since they only represent the direction of flow.

D. APPLICATION OF THESIS

Concept-Flow Diagrams are extremely useful and immediately applicable to CAI and computer tutorials. They benefit both the software designer and the user. Software designers have at their disposal a structured analysis design tool to improve the presentation and flow of their program. Also, they are given a tool that looks somewhat familiar to software engineers, making the learning of CFD design easier. The user, on the other hand, is able to use a CAI program that better suits his/her needs. The tutorial is more useful to the general user. Through the use of Concept-Flow Diagrams, the novice user is challenged but not frustrated with the presentation of new material and the experienced user is allowed to demonstrate understanding rather than becoming bored with presentation of concepts he/she already comprehends. Help is provided only when the user demonstrates a deficiency, either through task performance or explicitly. Concept-Flow Diagrams, thus, improve the functionality of CAI tutorials.

E. THESIS CONTENTS

Chapter II explains the methodology behind the CFD. Chapter III explains the application of the CFD methodology to the design of an actual tutorial that will be implemented. Chapter IV describes the program that was written for the tutorial implementation. And finally, chapter V presents the conclusions and directions for future research.

II. CONCEPT-FLOW DIAGRAM METHODOLOGY

A. PHYSICS OF SOUND TUTORIAL

The Concept-Flow Diagram (CFD) design methodology was developed to aid in the design of a tutorial concerning the physics of underwater sound. This area was chosen because the U. S. Navy, specifically, Space and Naval Warfare Systems Command (SPAWARS (PMW-183)), requested that we design a tutorial to present just that. The intended users of the tutorial are U. S. Navy enlisted personnel in the Ocean Systems Technician Analyst (OTA) rating. This user group includes the high school graduate who basically has no knowledge of the topic, and the experienced OTA, who has worked with and studied the physics of sound. The tutorial begins with an introduction of how to use the particular computer program. The concepts covered in the tutorial are basic definitions regarding the physics of sound, the characteristics of sound, ocean characteristics, ray path transmission and loss, the passive sonar equation, and the sound velocity profile. The placement of mastery tests (MT) and the concept-flow of the Physics of Sound Tutorial (POST) are discussed in this and the following chapter.

B. PEDAGOGICAL PHASE OF DESIGN

"The key to pedagogical design, in all its phases, is the extremely good teachers. It is the competence of the good teacher that one tries to capture within the computer program." (Bork, 1990, p. 6) As suggested by Bork's methodology (Bork, p. 106), a professor of physics was consulted for the pedagogical design phase of the POST. To form a pedagogical basis for the design, designers must determine two things to allow for easy transition to the design of the CFD. First, the pedagogical phase must determine exactly what the concept goals are and second, what tasks to include in the MT. These determinations are formed in a series of four steps.

1. User Group

The first step is to determine the user or person/group of people for whom the system is being built.

> The user may be job specific such as an operator, a supervisor or an executive; the user may be based on the level of experience of the potential users of the system; or the user may be inherent in the concepts being taught. (Yourdon, 1989, pp. 155-156)

Knowing who the users of the tutorial will be affects the pedagogy of the tutorial. The terminology, complexity of mastery tests, and the order of the presentation are much different for the novice than the expert user. The pedagogy for the novice user begins with low-level concept goals and mastery tests and increases in complexity as the user completes portions of the tutorial. The expert user, on the other hand, begins with high-level concept goals and tasks.

2. Concept Goals

Once the user group is identified, the next step is to examine the tutorial system as a whole and determine the overall concept goals. In other words, the analyst needs to decide what the student must comprehend upon completion of the tutorial. This step can take the form of a list of all of the concepts needed for presentation in some form within the tutorial. This list of concepts is then grouped into categories of related concepts and each category assigned a descriptive name. In the case of the POST, we produced the list in Figure 1.

3. Concept Dependencies

Once the concept goals are known, the next step is to determine the dependency relationship between each of the concept categories. In other words, the analyst needs to decide what concepts must be taught prior to other concepts in order to facilitate comprehension. For example, it makes no sense to present the passive sonar equation before presenting what a source, sound and detector are and what the relationship and behaviors are in the medium through which the sound is travelling. As the dependency relationships became apparent, the order of presentation of these groups based on the dependencies also became apparent.

The order of the presentation is not linear. Linearity occurs when the entire tutorial presentation is restricted to the presentation of one concept after another in a specific order. While this may be appropriate for a fully homogenous group of students, it forces experts to review known material. If the order is initially linear, subdividing and rearranging the contents of the concept goal categories is necessary until the dependencies are more explicit. The subdivision and rearrangement of the concept goal categories leads to the formation of concept categories that allow the user's knowledge to derive the order of presentation within a section of the tutorial. The

Introduction and Basic Definitions
- Source
- Medium
- Sound and Ray Path
- Detector

Sound Characteristics
- Frequency
- Hertz
- Period
- Compression
- Rarefaction
- Longitudinal wave
- Broadband
- Amplitude
- Effective Pressure Amplitude
- Wavelength
- Wavefront
- Absolute Sound Pressure Level
- Decibel
- Tonals

Ray Path Transmission and Loss
- Attenuation and Absorption
- Spreading/divergence
- Spherical Spreading
- Cylindrical Spreading
- Scattering
- Critical Angle
- Multipath Propagation
- Directed Path
- Reflected Path
- Refracted Surface Reflected Path
- Refracted Path
- Limiting Ray
- Shadow Zone

Ocean Characteristics
- Bathymetry
- Isothermal
- Gradient
- Thermocline
- Sound Channel
- Surface Ducts
- Deep Sound Channel
- Convergence Zone
- Reliable Acoustic Path
- Noise
- Biological Noise
- Hydrodynamic Noise
- Ocean Traffic Noise
- Sea Surface Noise
- Seismic Noise
- Bottom Bounce
- Mixed Layer

Sound Velocity Profile
- Temperature
- Salinity
- Pressure
- Deep Sound Channel Axis

Passive Sonar Equation
- Transmission Loss
- Noise Level
- Recognition Differential
- Signal Excess
- Bandwidth
- Echo Level
- Source Level
- Array Gain
- Figure of Merit
- Noise Spectrum Level
- Doppler

FIGURE 1: Concept Categories

original list of concept goal categories and their dependency relationship for POST is illustrated in Figure 2. The original order of the concept presentation is linear. Upon reexamination of the concept categories and their contents, we discovered that several of the categories were related and could be combined. The reexamination included looking for related concepts, no matter which category they were a part. In Figure 2, *Ray Path Transmission* and *Transmission and Propagation Loss* categories are related, so these two categories were combined. The rearrangement of the categories changed the dependency relationship. Categories emerged whose order of presentation would be left to the student. Students may choose if *Ray Path Transmission and Loss* is presented before or after *Ocean Characteristics*. Figure 3 illustrates the refined concept dependency relationships.

4. Mastery Test Placement

The last step is to determine where in the system to place the mastery tests and what tasks to include within these mastery tests. A MT is a task or a group of tasks such that the tasks are "... large enough to expose the student's misunderstandings and correct them." (Walker, 1984, p. 43) User performance upon completion of the task determines the next state in the tutorial flow. If the user successfully completes the task, the MT directs the tutorial flow in a forward direction advancing to another state or concept. However, if the user does not successfully complete the task, answer analysis within the MT determines the tutorial flow.

Answer analysis of a task consists of determining the possible answers, both correct and incorrect, and if necessary, the types of answers given to previous tasks. By maintaining a history of previous answers, a pattern of errors made may develop making the area of deficiency more specific. If answer analysis cannot immediately determine the exact area of deficiency, the tutorial backtracks through the mastery tests until the deficient area is pinpointed. The mastery tests, not the concept bubbles, determine the user's level of comprehension.

Mastery tests are a part of the bubble to ensure the user comprehends the low-level concept goals. Mastery tests can and should be given upon completion of one or more related bubbles to ensure the user is not only grasping the individual concepts goals, but is also able to tie these concepts together to understand the more general concept goal. The placement of mastery tests is illustrated in Figure 4. Successive tasks within a MT should increase in complexity. By gradually increasing the complexity of each task, the MT design makes answer analysis easier.

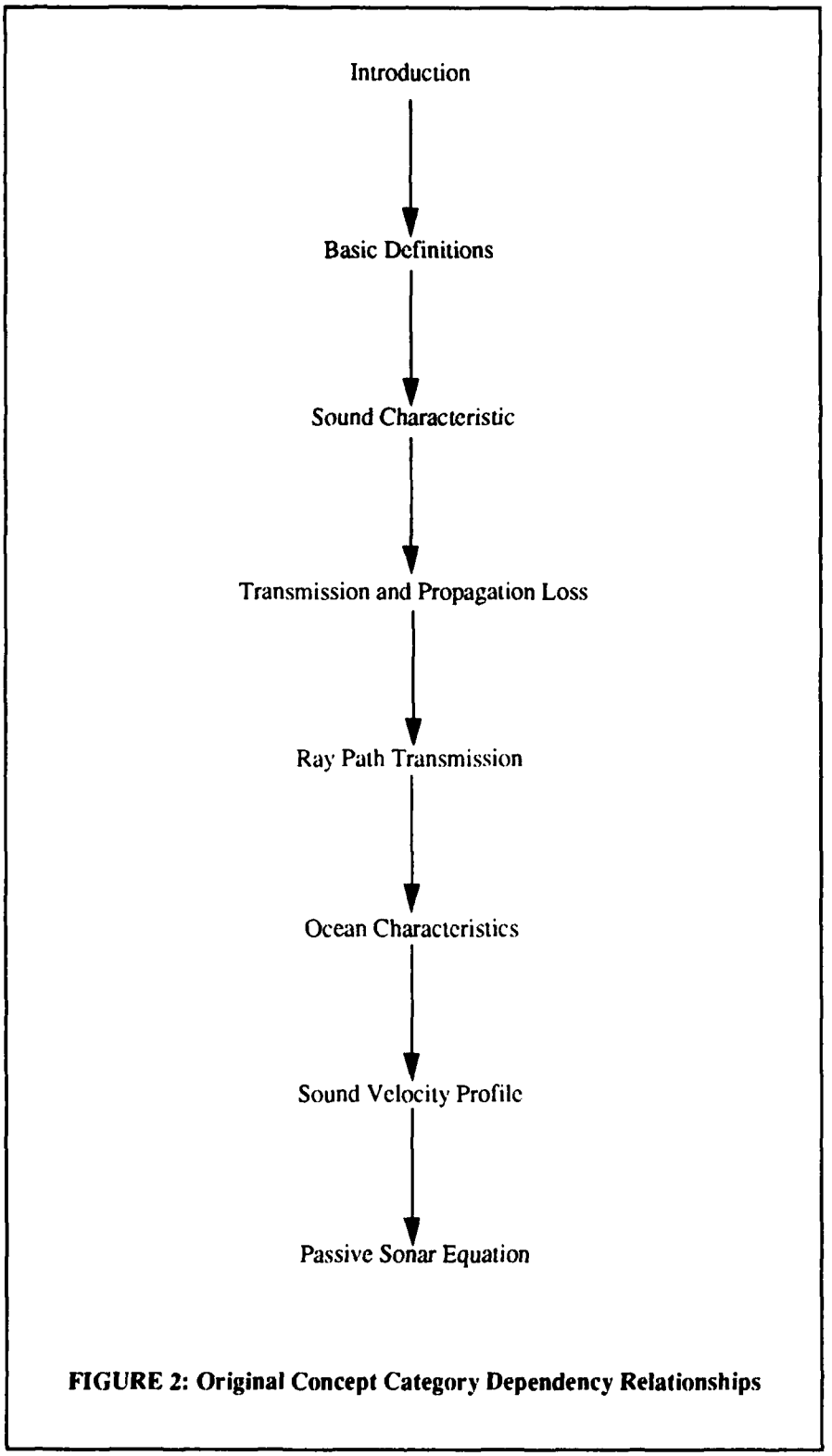

FIGURE 2: Original Concept Category Dependency Relationships

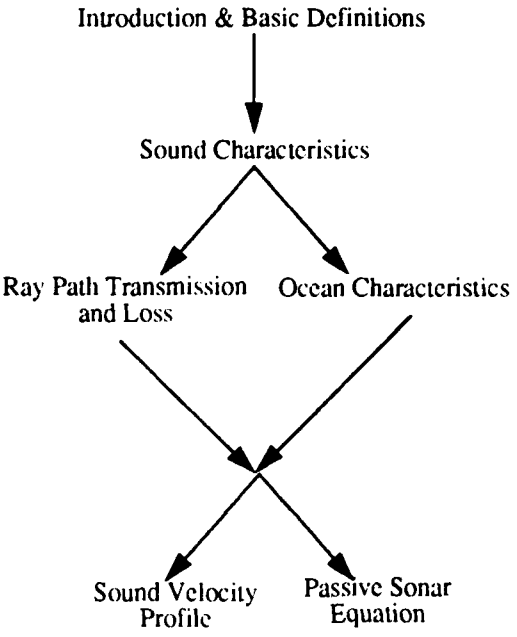

FIGURE 3: Refined Concept Category Dependency Relationships

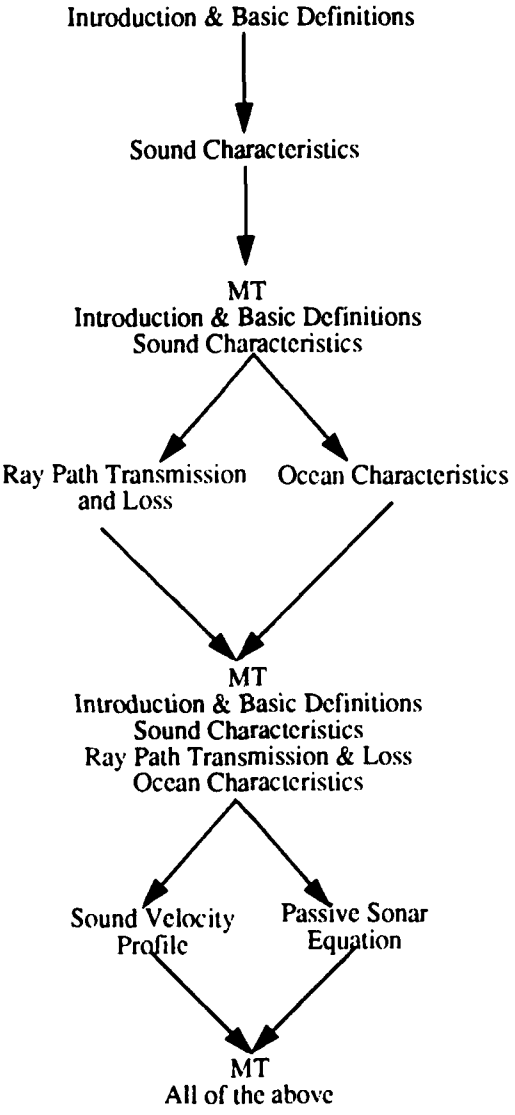

FIGURE 4: MT Placement

C. CONCEPT-FLOW DIAGRAM

Upon completion of the pedagogical phase of design, the next step is to formulate the concept-flow by using the CFD. At this point in the tutorial design process, the concepts to be presented and the tasks to include in the MT have been determined in the pedagogical phase. The basic premise behind the CFD is to let the user demonstrate level of comprehension and delve into the actual concept presentation only if necessary, i.e., the user demonstrates a deficiency in a specific area. The user should have the option of either starting with a tutorial presentation or delving into a MT. Theoretically, the expert user should be able to traverse through the tutorial from MT to MT without a concept presentation occurring.

The CFD design consists of levels as with the Dataflow Diagram (DFD). The top level of the CFD is level 1, the second level is level 2, and so on. The high-level diagrams will have small numbers and the low-level diagrams will have large numbers. The higher levels, i.e., level 1, contain general concept goals and mastery tests; the low levels contain the specific concept goals and mastery tests.

1. Concept Bubble

The concept categories delineated in the pedagogical design phase translate into the high-level bubbles in the CFD. The dependency relationships determined in the pedagogical design phase translate into the placement of the bubbles and the arrows between them to indicate flow of the tutorial presentation. The contents of each bubble becomes more specific as you move down (levels with high numbers) in the CFD, until the lowest level is reached. The concept goals within each concept category of the pedagogical design are translated into the lowest level of the CFD. The lowest level of the CFD pinpoints the exact concepts to be presented and the order of presentation. It is acceptable for the lowest level concept order of presentation to be linear.

There should be no more than 9 concepts per level, including level 1. "People can deal with seven, plus or minus two, chunks of information consciously and comfortable. More than nine chunks of information can lead to confusion and overload." (Cleveland, 1986, p. 18) If more than 9 concepts per level occur, the pedagogical design must be reviewed to determine if concept categories can be combined at the higher levels then broken out at the lower levels. Each category must contain closely related concepts, not just an arbitrary group of concepts. Therefore, a concept category may be divided into subcategories. The subcategories then translate to intermediate levels of the CFD and the specific concepts are the lowest level of the CFD.

Each bubble is numbered as with the DFD. Numbering each bubble allows for relating a bubble to the surrounding levels of Concept-Flow Diagrams. The number of each low-level CFD relates to the high-

level CFD bubbles. For example, bubble 1.1.1 is a level 3 diagram and the bubble is part of bubble 1.1 in level 2 and bubble 1 in level 1. Reversing the process, if bubble 1 in level 1 of the CFD is broken down, the level 2 bubbles will be numbered 1.1, 1.2, 1.3, etc.

2. Mastery Test Octagon

This is probably the most difficult determination to be made in the design of the tutorial. The mastery tests determine the level of comprehension of the user and whether or not the user is ready to move forward or backward in the flow of the tutorial. The MT is therefore extremely important in the design of the tutorial.

Great care must be taken in determining the placement of the mastery tests and determining the tasks presented within each MT. This does not necessarily mean that in the higher level CFD, i.e., level 1, there must be a MT given after each concept. When designing POST, the first attempt at MT placement put at MT between each and every bubble. We discovered that this is unnecessary. A MT between two bubbles is unnecessary if the flow to the second bubble is only dependent upon the previous bubble. For example, it is unnecessary to place a MT between the *Introduction & Basic Definitions* bubble and the Sound Characteristics bubble in Figure 3. The MT within the *Introduction & Basic Definitions* bubble is sufficient to determine user comprehension of that concept category. Placing another MT between the bubbles would only be redundant. It is necessary, however, to place a MT after the completion of the *Sound Characteristics* bubble because the user must comprehend both the *Introduction & Basic Definitions* bubble and the *Sound Characteristics* bubble prior to starting either the *Ray Path Transmission and Loss* bubble or the *Ocean Characteristics* bubble.

There are smaller mastery tests given within each bubble in order to determine comprehension of that particular concept. The mastery tests within a bubble address specific concepts within that bubble. At the end of the presentation of a bubble and prior to flowing to the next bubble, there is a MT to ensure that the user comprehends the concept category presented. Then, if a MT follows in the flow, it tests comprehension of the relationship between that bubble and any other bubbles already presented. For example, in Figure 4, the mastery tests within the *Introduction & Basic Definitions* bubble and the *Sound Characteristics* bubble test only their respective concepts. The MT given after the *Sound Characteristics* bubble tests the comprehension of the relationship between the *Basic Definitions* and the *Sound Characteristics.*

The addition of the MT to the diagram complicates the 9 bubbles per level rule. If the addition of mastery tests clutters the diagram, the entire CFD must be readdressed. It may require that more concepts that

are related be combined and/or the placement of mastery tests changed. Mastery tests are numbered just as with the bubbles.

3. Guidelines

This translation into a CFD is by no means the final CFD. The CFD must constantly be reviewed for improvement. The following guidelines and criteria must be considered:

1. Are there more than 9 bubbles on each level of the CFD? The CFD cannot look too busy or cluttered. If there are more than this number of bubbles and mastery tests, take another look at the concept categories determined in the pedagogical design. Concept categories may need to be combined or reorganized to meet this criteria. Divide the CFD into more levels than planned or reevaluate the use and placement of mastery tests. This only leads to better modularization of, flow of and answer analysis within the tutorial.

2. Do the mastery tests serve as a point of bottleneck? A bottleneck occurs when the MT is testing too broad and area for comprehension. If so, reconsider the mastery tests that are needed in the tutorial. A bottleneck indicates a need for further breakdown of concept categories and placement of mastery tests within the tutorial.

3. Does the design allow backtracking from MT to MT?

4. Does the design allow the user to choose between concept presentation or MT presentation?

III. CONCEPT-FLOW DIAGRAM METHODOLOGY AS APPLIED TO THE PHYSICS OF SOUND TUTORIAL

The pedagogical design phase of the Physics of Sound Tutorial (POST) has been completed as discussed in the previous chapter. The concept categories and their subgoals are now organized to be incorporated into the tutorial by using the Concept-Flow Diagram (CFD). The following discussion applies the CFD and the Dataflow Diagram (DFD) methodologies to POST.

A. CONCEPT-FLOW DIAGRAM

1. Level 1

Level 1 of the CFD is illustrated in Figure 5. At this level, the CFD shows the general concept categories introduced and when the mastery tests occur. The user may enter at one of two points: go directly into the MT or start with the concept presentation. The bubbles are the concept categories determined during the pedagogical design phase (Figure 1). The flow and MT placement correspond to the relationships determined during the pedagogical design phase (Figure 4). The mastery tests visually break up the tutorial into three distinct areas: bubbles 1 and 2; bubbles 3 and 4; and bubbles 5 and 6. This was not intentional; the thought that went into the pedagogical design and the final dependency relationships made this a natural flow.

The contents of the *Introduction and Basic Definitions* bubble in Figure 5 are listed below. The other concepts will be implemented in the future and therefore, the specific areas covered within each topic are not discussed.

1. Buttonology - how the mouse operates and how each button that appears on every screen operates. Buttons introduced are HELP, CONTINUE and EXIT.

2. Source - the definition of the term, how represented in the tutorial and manipulation of the source by the user.

3. Medium - the definition of the term and how the medium affects the speed of sound.

4. Sound and Ray Path - the definition of the term and how represented in the tutorial.

5. Detector - the definition of the term, how represented in the tutorial, manipulation of the detector by the user and the effects on detection of sound depending upon the relationship between the source and the detector.

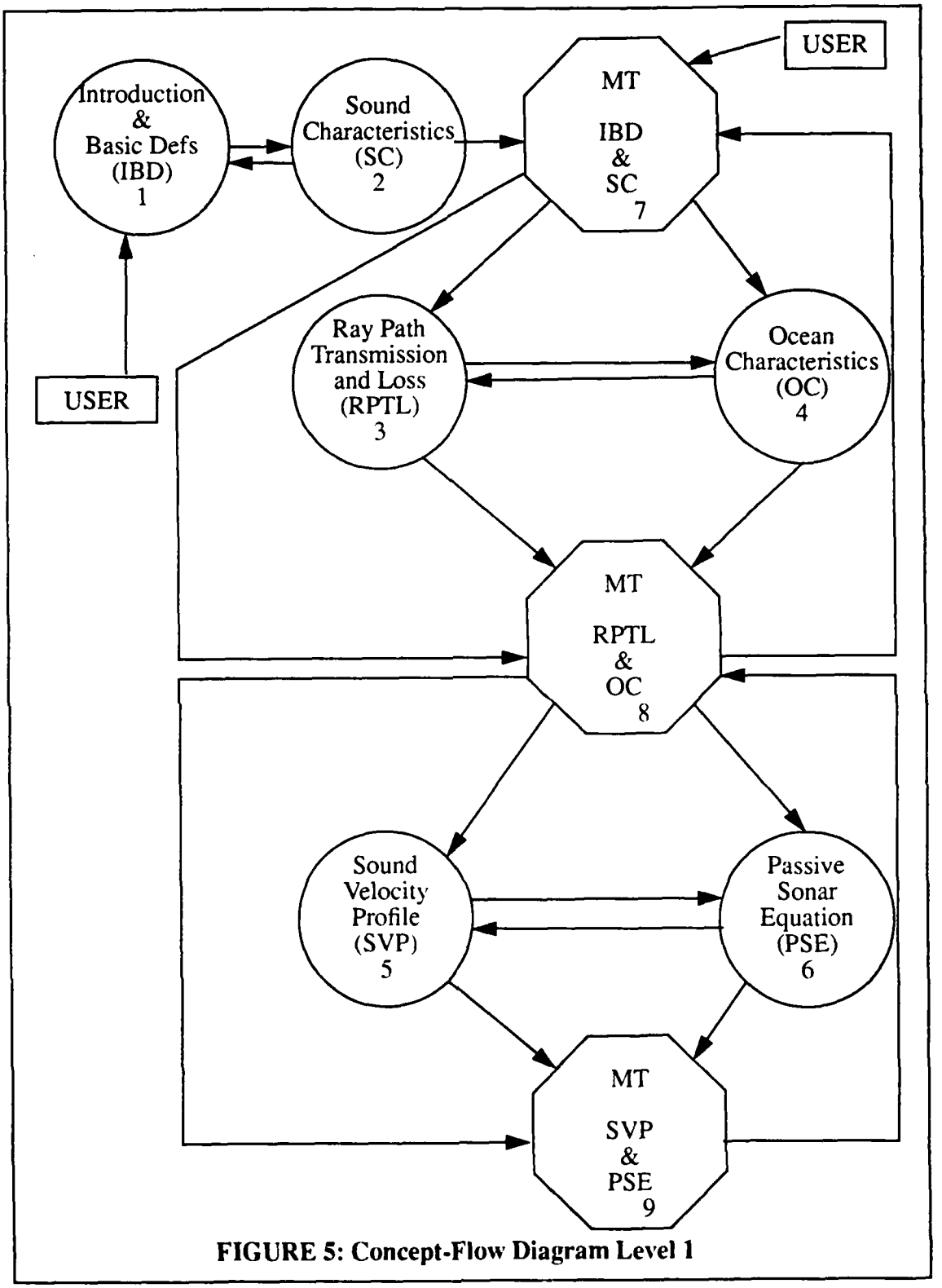

FIGURE 5: Concept-Flow Diagram Level 1

2. Level 2

Refer to Figure 6. he first attempt at designing the level 2 diagram is illustrated in Figure 7. Notice there is a MT after each concept presentation. Upon further design and review of the pedagogical design, we discovered that this was unnecessary for the POST. The concepts presented are so basic that the MT for each concept would be trivial. A MT within and between each concept bubble caused the MT between each concept bubble to be trivial and redundant. We decided that a MT for concepts 1.2 thru 1.5 was much more effective as illustrated in Figure 7.

Each bubble in Figure 7 represents the presentation of each concept. Bubbles 1.2 thru 1.5 are not contained within one bubble because each concept presentation depends upon the student understanding the prior concept presentation. Although there are not explicit mastery tests between each bubble, there are mastery tests within each bubble. It is the successful completion of these mastery tests that determines transition to the next bubble.

3. Level 3

The original version of the CFD level 3 diagram appears in Figure 8. Notice that there are no mastery tests. Upon reexamination, we decided that the *Introduction* bubble needed to be more explicit and that mastery tests were needed. The inclusion of mastery tests ensured that the user had mastered the basic tasks of using the mouse prior to starting the tutorial. This diagram was revised, Figure 9, to include them. Mastery tests are not given after the presentation of each concept goal because the MT after all of the concepts at this level have been presented would be too trivial and redundant. The concepts themselves are so basic that one MT at the end is sufficient.

B. DATAFLOW DIAGRAM

1. Context Diagram

The context diagram represents the entire system. The context diagram for POST is illustrated in Figure 10. In addition to the student using the tutorial, terminators are introduced representing an instructor and the Ocean Systems Qualification (OQS) board. The instructor and the OQS board are able to monitor student progress and to update the tutorial as needed.

2. Level 1

Level 1 of the DFD is illustrated in Figure 11. A quick glance at Figures5 and 11 shows a drastic difference in the CFD and the DFD. The difference is due to what each analysis tool represents. The DFD

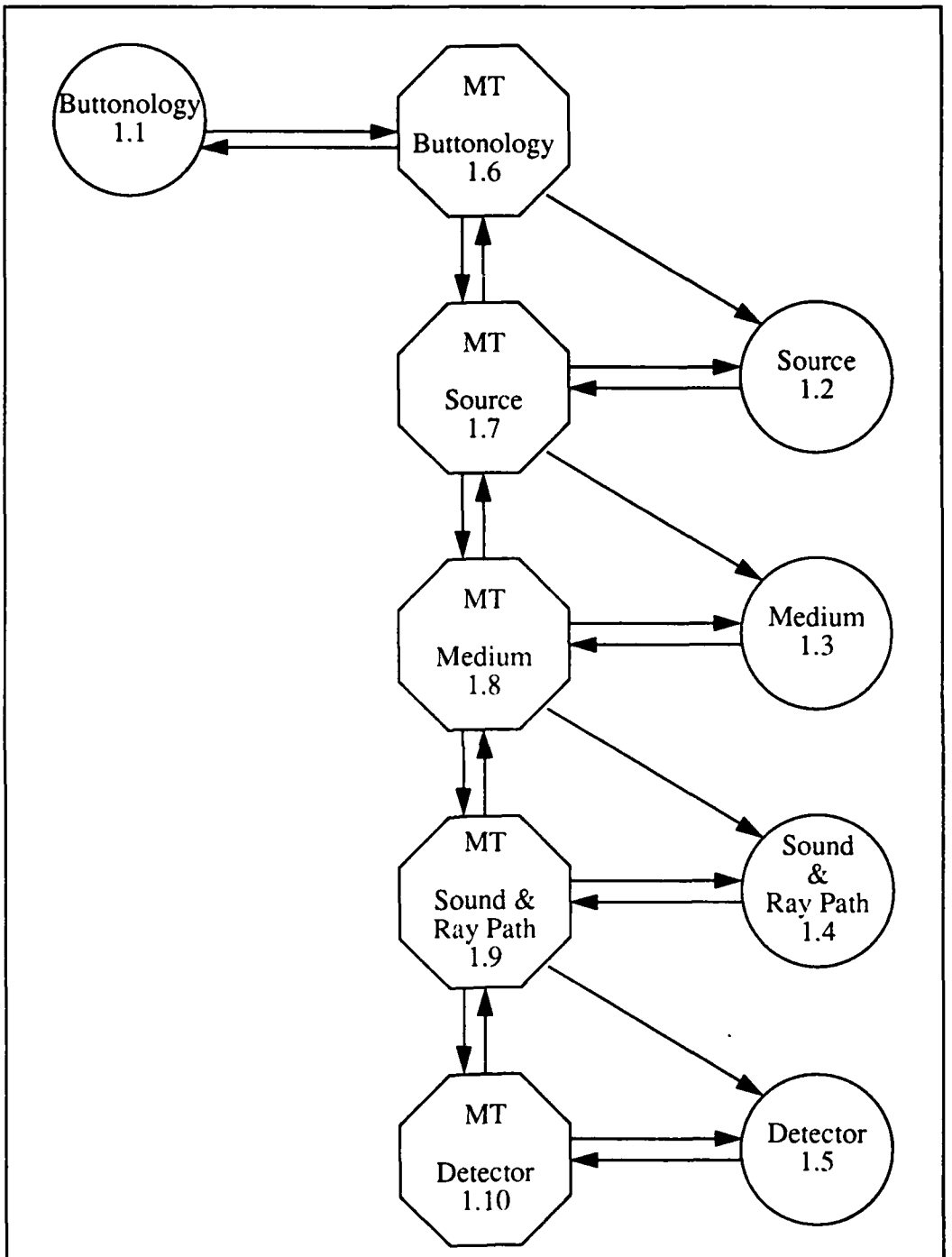

FIGURE 6: Original Concept-Flow Diagram Level 2

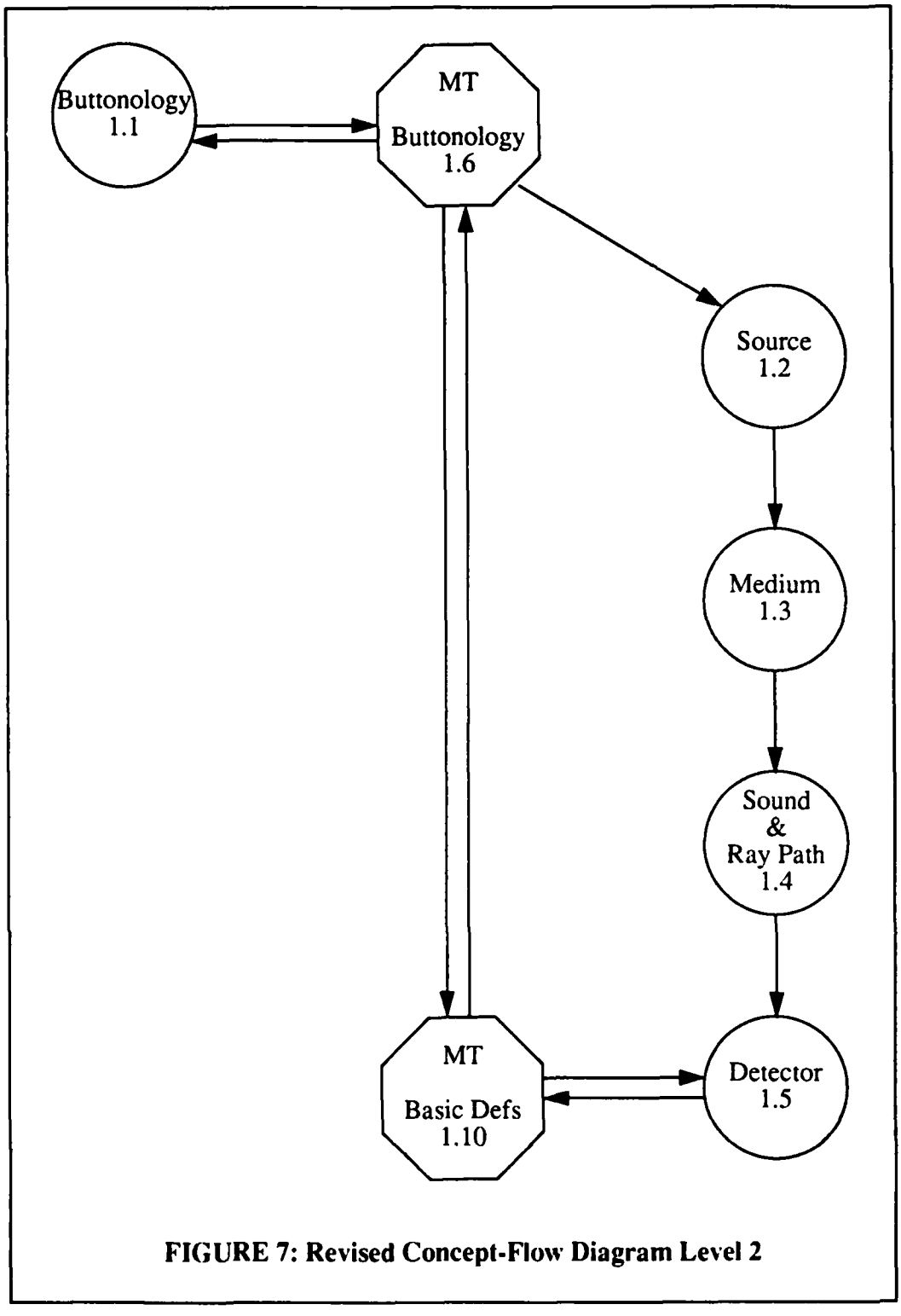

FIGURE 7: Revised Concept-Flow Diagram Level 2

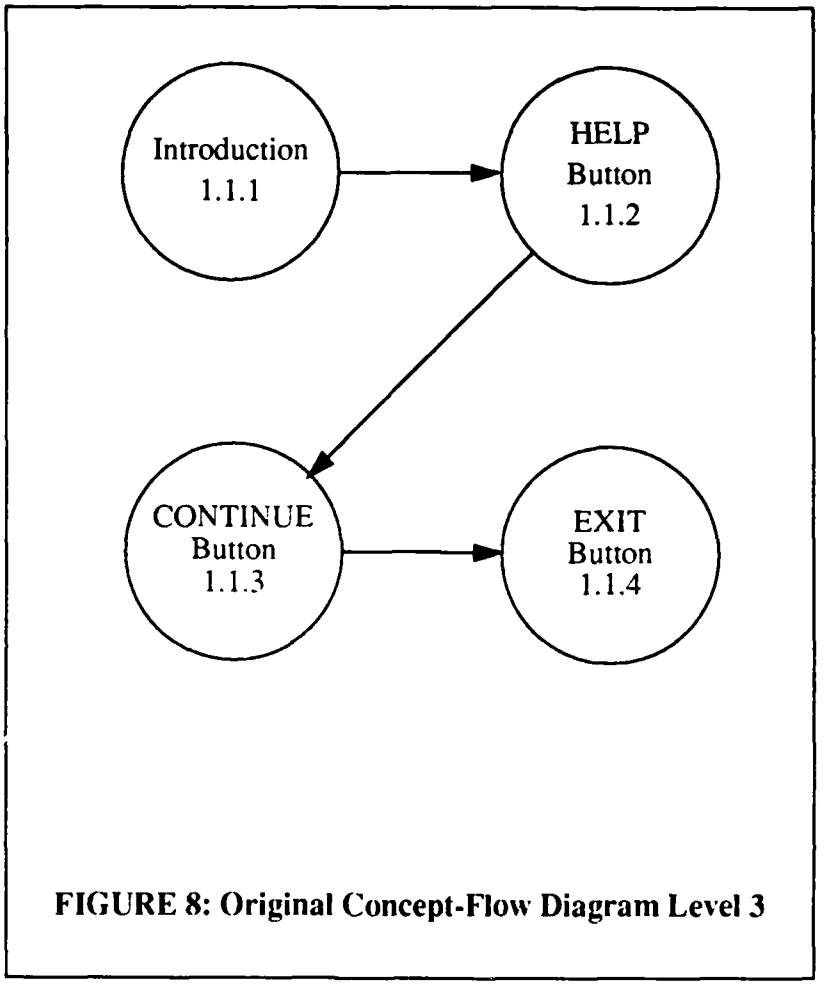

FIGURE 8: Original Concept-Flow Diagram Level 3

represents the communication between software structures of the system and shows how the data is passed throughout the system. The CFD attempts to provide a visual aid to illustrate how the tutorial is to be presented; i.e., the sequence of introduction of different concepts.

The terminators of the POST tutorial are:

1. The instructor. The instructor is allowed to interact with the tutorial text and test question file. Interaction includes the instructor deciding which concept module the student will use or modifying the test questions asked of the student. The instructor also may keep track of user mastery test (MT) results.

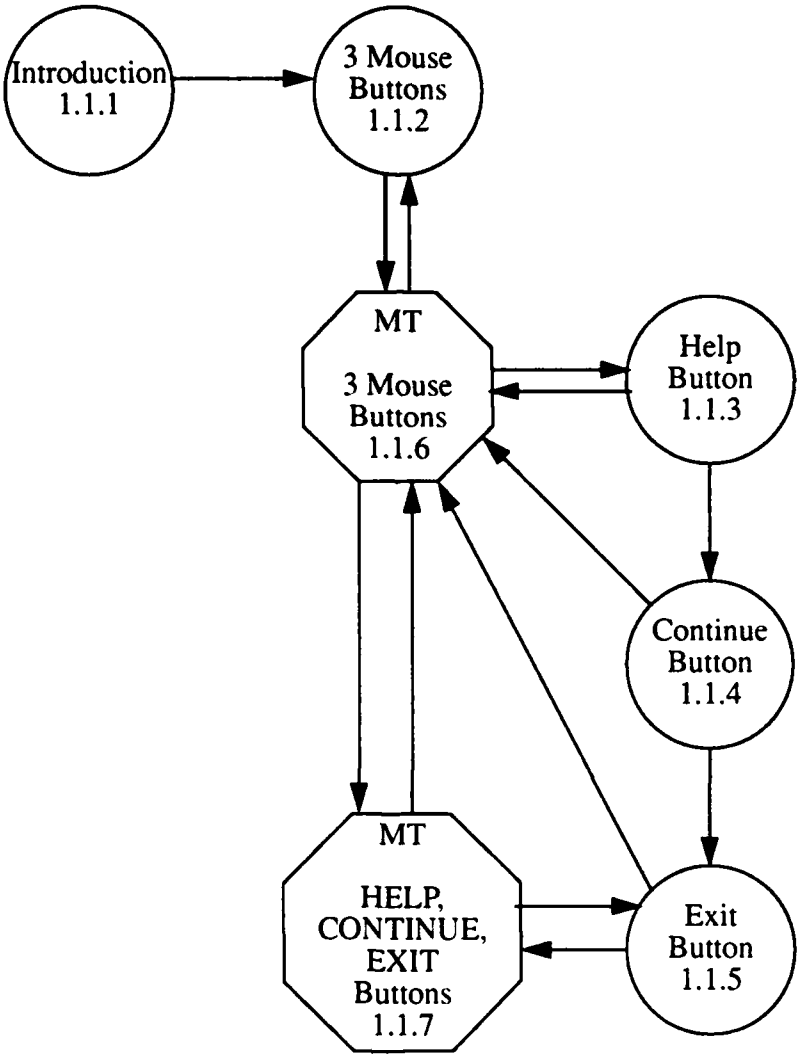

FIGURE 9: Revised Concept-Flow Diagram Level 3

2. The Ocean System Qualification Standard (OQS) Board. The OQS Board may interact with the system by requesting and/or monitoring MT results of each user.

3. The user of the tutorial. The user may request which concept he/she wishes to explore. If the user must quit the tutorial, either temporarily as an icon or for an extended period of time, he/she may restart the tutorial at the point at which he/she quit. Determination of this point of restart is through check points reached when the user last used the tutorial.

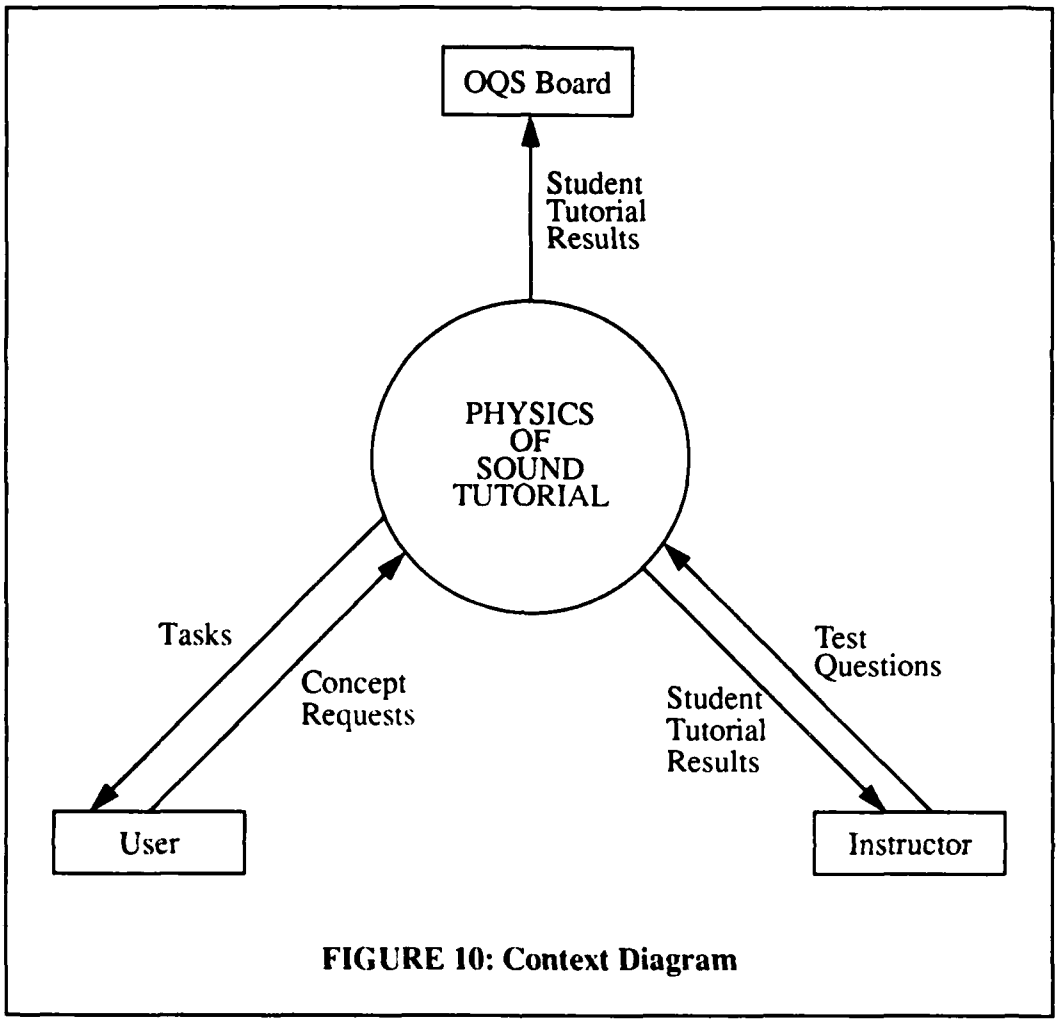

FIGURE 10: Context Diagram

Bubble 1 of Figure 11 allows the instructor to interact with the tutorial text and test questions file. The tutorial presentation uses this file for the text of concept explanations and mastery tests. For each tutorial, the main process is to present the tutorial, bubble 2. The CFD is actually a more detailed design of this bubble. It delineates exactly how to present the tutorial, taking into consideration the topic of the lesson plan. Because the mastery tests are given within this process, the instructor and the OQS board get the individual user performance statistics from here. The other process that occurs in a tutorial is the restart, bubble 3. If the user exits the tutorial in the middle, the user is brought back to the concept where he/she left off. The tutorial has check points assigned after the completion of a concept or MT. When the user restarts the tutorial, the check point file is checked find the last check point encountered. The tutorial then restarts from this check point.

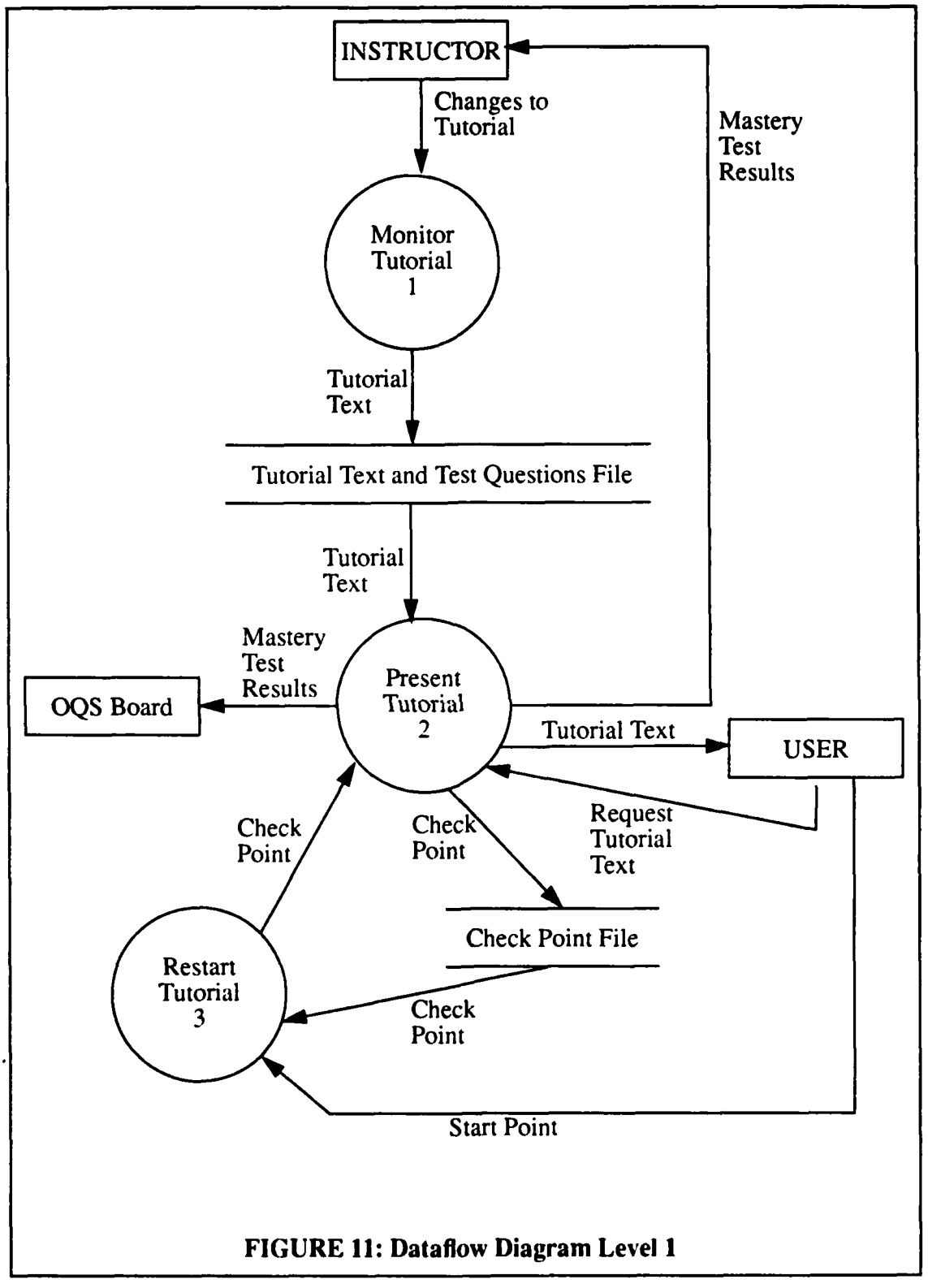

FIGURE 11: Dataflow Diagram Level 1

To illustrate the basic process flow, the only DFD level 1 bubble that required further breakdown was bubble 2, Present Concept. Refer to Figure 12. Within the presentation of the tutorial, the two major processes that occur are either the presentation of a concept to the user or the presentation of a MT to the user.

1. Once the concept presentation occurs, a MT is given to determine the level of comprehension of the user. Determination is made by the maintenance of statistics of correct answers to the mastery tests.

2. Once the MT is given, a determination is made by the tutorial as to the level of comprehension. If the user has shown mastery of the concept, the tutorial moves on to the next MT or concept module. If the user has shown a deficiency, the tutorial moves to the appropriate concept presentation to help clear the misunderstanding.

C. SUMMARY OF CFD AND DFD COMPARISON

The CFD design methodology presents a very complex structure while the DFD presents a simple structure. This is very characteristic of computer-aided instruction (CAI). CAI deals with the trying to present a wide range of concepts in an ordered manner. The CFD structure allows for the breaking up of this task into an organized hierarchical structure.

The CFD allows for an instructor to interact with the tutorial presentation to modify task and concept discussion. This permits the tutorial to be updated as the course develops. By allowing the modification of the tutorial, the CFD structure must be placed in an external file, interpreted by a fairly general system. This system is presented in the next chapter.

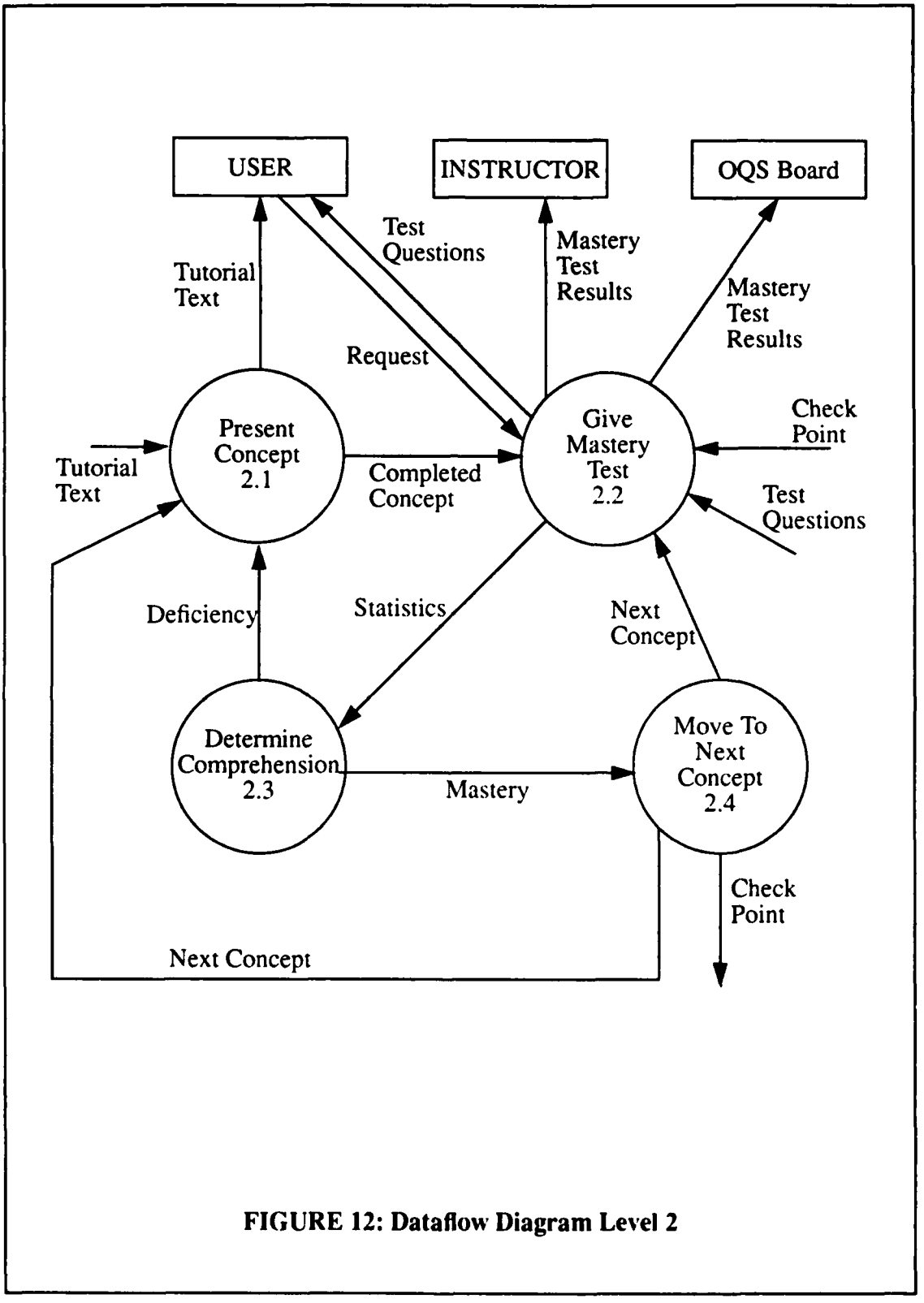

FIGURE 12: Dataflow Diagram Level 2

IV. DESCRIPTION OF PROGRAM

Now that the structure of the Concept-Flow Diagram (CFD) has been developed, it is necessary to translate the design into software. In order to do this translation in a flexible manner to allow instructors to customize the course materials, the CFD is translated into an external file in a special-purpose language. This language is parsed into a data structure representation and interpreted. The following discussion presents the grammar implemented to support the data structure and a prototype of an interpreter.

A. COMPUTER SYSTEM

The Physics of Sound Tutorial (POST) module is part of the SPARS Release 5 system. The computer hardware used for implementation was a U. S. Navy Standard Desk-Top Tactical-Support Computer (DTC-2) designed by a major systems integration firm, C3, Inc. The DTC-2 uses the SPARC 4 series implementation of RISC computer architecture. The DTC-2 system includes an 8 MB 4/110 CPU, a 19" color monitor, a color graphics plotter, a color graphics printer, a mouse, and a track ball. The software used was SUNOS 4.0 and C compiler. The 4/110 UNIX System V Operating System includes SunView, Open Windows (X11/NeWS), NFS, Assembler, and Real-time Extensions.

The interpreter was implemented using Sun Visual/Integrated Environment for Workstations (SunView). SunView is a tool that allows for the implementation of graphic-based applications running in windows. Two types of windows were used in this application: panels in order to use buttons and a canvas in order to draw text and graphics. The canvas may be used as a whole region or a set of nine regions. A mouse is used to track location, to click the set of buttons, and as a graphic positioning device. A trackball is available for use but was not used in the prototype.

B. IMPLEMENTATION OF DESIGN TOOLS

We developed a data structure, a grammar and an interpreter in order to implement the tutorial based upon the pedagogical design phase, structured analysis and CFD designs. The POST was implemented using the grammar and data structures described below. Appendices B thru E contain the POST script, grammar, LEX and YACC files and interpreter respectively. Appendix F contains a detailed explanation of the data structure elements. In the discussions below, bold italics refer to elements of the ***grammar*** and bold refers to elements of the **data structure.**

C. DESCRIPTION OF GRAMMAR AND DATA STRUCTURE

The grammar is based on the CFD graph semantics and expressed in a BNF notation. The UNIX tools LEX and YACC were used to implement the grammar as a parse for input files describing the tutorial. The data structure for a tutorial forms a ***cfd_graph***. A ***cfd_graph*** consists of a ***cfd_menu*** and nodes, called ***cfd_node***. The reserved words of the grammar are listed in Figure 13.

assert	halfwid	(
clear	input	)
click-left	key	+, -
click-mid	mouse	*, /
click-right	mouse&key	==
click-any	mouse-move	<, >
click-help	past	<=, >=
click-continue	pause	& (logical and)
click-exit	write	| (logical or)
draw	.x	
halfht	.y	

FIGURE 13: Reserved Words

1. Graphic Definition

The user may define the graphics or CFD states at the highest syntax level of the input file. The graphics are specified by ***identifier := string***. The tutorial script references the graphic images by the ***identifier***. The ***string*** identifies a file in which the graphic is stored. This information is used in the draw and drag actions. Figure 14 shows the graphic image definitions used in POST *Introduction and Basic Definitions*.

2. cfd_node

Each node of the CFD is represented as a ***cfd_node*** structure with a CFD node identification number (***cfd_id***), the **action** or actions that is/are to occur at this CFD node, and the possible response or

```
detector := "detector_sym.icon
mouse_sym := "mouse_sym.icon"
path := "path_sym.icon"
post := "post_sym.icon"
source := "source_sym.icon"
```

FIGURE 14: Graphic Definition Example

responses expected at this point in the tutorial. An example of the grammar of a typical ***cfd_node*** appears in Figure 15.

```
(st_1_1_1, ( (0, clear),
 (1, draw, post@(mouseX, mouseY)),
(2, write, "Welcome to the Physics of Sound Tutorial"),
(5, write, "Upon completion of this tutorial, you will have enough of an understanding to
complete the Physics of Sound module of your OQS"),
(8, write, "Let's begin . . ."),
(0, pause, 15) ),
st_1_1_5)
```

FIGURE 15: Example of a State

a. action_list

The action list specifies the actions that are to occur while in a state. The action list identifies where in the window to accept input, to draw graphic images and where to display the text of the tutorial. The **actlist** of the **cfdnode** in the data structure represents the action list elements as a linked list data structure of **actnode**. The **actnode** identifies the region of the window in which the action is to occur, the specific action to take, and the arguments to that action.

(1) ***region_id*** - The screen window is broken into 10 distinct regions. By breaking the window into distinct regions, different actions are allowed to take place in the different regions of the window. This allows the tutorial to simultaneously display graphics and text. In Figure 15, the "0" in (0, clear) refers to region 0, the "8" in (8, write, "Let's begin . . .") refers to region 8.

(2) ***draw*** - This terminal enables the tutorial to display a graphic in the window to enhance the presentation of the tutorial. The "***draw***" action must be followed by an ***identifier*** specifying the graphic to be displayed and the ***location***. The ***location*** is given using a set of coordinates; either the current location of the mouse, (***mouseX***, ***mouseY***), or the location of a graphic currently displayed, (***identifier.x***, ***identifier.y***). In Figure 15, (1, draw, post@(mouseX, mouseY)) says to draw the post icon in region 1 at the current location of the mouse.

(3) ***clear*** - This feature allows the erasure from portions of the window text and/or graphics that are not relevant to the current state of the tutorial presentation. The ***location*** is identified as in the "***draw***" action. In Figure 15, (0, clear) says to clear region 0.

(4) ***write*** - This is the means by which text is displayed in the window. In Figure 15, (2, write, "Welcome to the Physics of Sound Tutorial") says to write the string "Welcome to the Physics of Sound Tutorial" in region 2.

(5) ***input*** - Restricts the area of the window where the user may input text and/or manipulate graphics. Restricting the input area to a region within the window prevents the user from arbitrarily writing text or moving graphics around the window and thus interrupting the tutorial presentation. Input is either ***mouse***, ***keyboard*** or ***mouse&key***. In Figure 16, (0, input, mouse) says to allow input via the mouse in region 0 and (9, input, keyboard) says to allow keyboard input in region 9.

(6) ***pause*** - Limits the amount of time that an action is displayed to the user before moving on to the next action or state. An example is to display text in the window, allow a sufficient amount of time for the display to be read, and then move on to the next state. In Figure 15, (0, pause, 15) says to set the timer for 15 seconds and have region 0 in the wait state.

(7) ***drag*** - Allows the user to drag a graphic using the mouse. The drag feature may be incorporated into the mastery tests to increase the complexity of the tasks.

(8) ***quit*** - This feature is used to quit the tutorial upon execution of the last state of the tutorial. If MT performance and answer analysis indicate that the user is not grasping a concept, the tutorial

(st_1_1_50, ((7, write, "Don't be afraid to try pressing a button on the mouse."),

(0, input, mouse),

(9, input, keyboard)),

((15 seconds & (past st_1_1_50 wait_wait), st_1_1_55),

(15 seconds & (past st_1_1_10 wait), (assert, wait_wait), st_1_1_50),

(click-left, (assert, left), st_1_1_15),

(click-middle & (past st_1_1_40 2X_wrong), st_1_1_45),

(click-middle & (past st_1_1_35 wrong_ans), st_1_1_40),

(click-middle, (assert, mid), st_1_1_35),

(click-right & (past st_1_1_40 2X_wrong), st_1_1_45),

(click-right & (past st_1_1_35 wrong_ans), st_1_1_40),

(click-right, (assert, mid), st_1_1_35),

(("Help" | "I do not know) & (past st_1_1_10 help), st_1_1_45),

(("Help" | "I do not kr ow), (assert, help), st_1_1_65),

((past st_1_1_10 wrong_ans & past st_1_1_35 wrong_ans), st_1_1_45),

((assert, wrong_ans), st_1_1_35)))

FIGURE 16: input and response_list Example

terminates the presentation and directs the user to get assistance from the instructor. This terminal is not the same as the EXIT button. The EXIT button is used by the user to request tutorial termination. In Figure 17, ***quit*** is used to stop the tutorial to allow the user to get assistance from a human.

b. response_list

The response list evaluates the user response, performs answer analysis, and provides the user with feedback. The response list was designed such that the tutorial can identify the user response and the state to flow to based upon the response and answer analysis.

The response list is a list of all of the possible responses to the action(s) in the ***action_list***. The structure of the response list must identify the response and the next state (***cfd_id***) to go to based on this

(st_1_1_45, ((3, clear),

(5, clear),

(3, write, "You seem to be having trouble."),

(4, write, "Before progressing any further, get assistance from your instructor."),

(0, quit)),

st_1_1_45)

FIGURE 17: quit Example

response. The response list is order dependent; therefore, the first response in the list that matches the response of the user is executed. The response list was implemented with a linked list called **resnode**.

(1) ***click-left, click-middle, click right, click-any*** - Refers to the respective button on the mouse. The click of a mouse button is a response event from the user. This terminal allows the response list to evaluate the respective response in the response list. In Figure 16, the "click-left" in (click-left, (assert, left), st_1_1_15) asks if the response event was a click of the left mouse button.

(2) ***click-help, click-continue or click-exit*** - Refers to the buttons located in the panel across the top of the window. The HELP button is on-line help provided to the user. The CONTINUE button is an accept action that directs the tutorial to move on to the next state in the tutorial flow. The EXIT button quits the tutorial. These buttons are always present in the window and are global to the tutorial allowing the user to choose one of these options at any time.

(3) ***mouse-move*** - Notifies the response list that the cursor is being moved using the mouse. This feature was not used in the POST.

(4) ***integer "seconds"*** - Limits the length of time that a user may take to give a response. This feature prevents the tutorial and the user from getting into a dead lock state; the user does not know the proper response and therefore is not giving any response and the tutorial is waiting for a response before moving on to the next state. In Figure 16, "15 seconds" in (15 seconds & (past st_1_1_50 wait_wait), st_1_1_55) states to set the timer for 15 seconds and wait that long for a response event. If a response event

is not received in that time period, go to state st_1_1_55. Remember, st_1_1_55 represents state 55 of concept bubble or MT 1.1.

(5) arithmetic operators - The use of the arithmetic operators allows for the interpretation of where a graphic has been placed within a region. This includes the use of the terminals ***halfht*** and ***halfwid*** to assist in centering a graphic in a region of the window.

c. Use of assert and past

In order to determine which state to go to next, answer analysis must occur. This involves reviewing answers given to past questions or tasks. Therefore, the MT should respond to a response differently based on a history of performance in the tutorial. ***assert*** is a list that maintains all of the states traversed and the user response for that state. However, the list does not imply path traversal. ***past*** verifies the assert list to check if a state and specified response have occurred in that state. The ***past*** feature is very useful for answer analysis. The choice of the next state to go to can then be based on the current response and previous responses made at specific states. This feature is also very useful for distinguishing state traversal for the expert versus the novice user. The novice user will make mistakes. The ***past*** feature allows for finding patterns in the types of wrong answers made and more importantly, ensures that the flow of the tutorial presentation is to a state that will provide assistance for correcting the pattern of wrong answers.

With the introduction of the ***past*** feature, the order of the response list affects the decision logic. For example, assume state st_1_1_10 has been traversed and st_1_1_10/wait exists in the assert list. Also, assume the response event is a click of the left mouse button. The evaluation of the response list in Figure 18 will always evaluate the first response and never evaluate the second response. The first response

```
(click-left & (past st_1_1_10 wait), (assert, wait_wait), st_1_1_70)
(click-left & (past st_1_1_70 wait_wait), st_1_1_55)
```

FIGURE 18: Sample Response List

will check the assert list to verify that st_1_1_10 has been traversed with the response wait. Since the response list is order dependent, this will always be true and the response will lead to an infinite loop. The evaluation of the response list in Figure 19, on the other hand, will evaluate the second response the first time the response list is encountered. The first time that state st_1_1_70 is traversed, only "st_1_1_10 wait" is in the assert list; therefore, "st_1_1_70 wait_wait" will not be found and the second response will be evaluated. The

(click-left & (past st_1_1_70 wait_wait), st_1_1_55)

(click-left & (past st_1_1_10 wait), (assert, wait_wait), st_1_1_70)

FIGURE 19: Revised Sample Response List

second time that state st_1_1_70 is traversed, both "st_1_1_10 wait" and "st_1_1_70 wait_wait" will be in the assert list. When the first response is evaluated, the response will be true and traversal will continue to another state. The terminals used to represent this data structure in the grammar are:

(1) ***past*** - Specify state and response given in that state. Response given cannot be a reserved word. Refer to Figure 13 for a listing of the reserved words. Refer to Figures 16, 18 and 19 for examples.

(2) ***assert*** - Specify response given which cannot be a reserved word. Refer to Figures 16, 18 and 19 for examples.

3. cfd_menu

This non-terminal and data structure element allows the tutorial to individualize the presentation by allowing the user to specify the point in the tutorial he/she wishes to start. If the user chooses a start point that is too advanced, the MT for that start point will identify this and backtrack until a point of comprehension is found. The interpreter of the tutorial script generates the window title of the specific tutorial in the main frame and the start points within the ***cfd_graph*** for the specific tutorial.

In the POST, the grammar for the menu appears in Figure 20. The main window will feature the string of the title "PHYSICS OF SOUND TUTORIAL" and the ***menulist*** consists of the title of the seven (7) choices to appear in the main menu and their respective state identifiers. The choices in the main menu will be labelled starting with the string "Introduction" and ending with the string "Overall Test."

D. DESCRIPTION OF INTERPRETER

An interpreter was designed to implement a prototype of the POST. The interpreter allows for the use of any tutorial written in the grammar described in this thesis. A listing of the interpreter is found in Appendix E.

Prior to interpreting the grammar, the grammar must first be parsed. The parse input is then used to initialize the tutorial window. The initial frame presents the user with the options for an initial start state. Once the initial start state is chosen, the actions of that start state are presented in the canvas of the base frame. The event handling procedures of SunView then accept a response from the user to be interpreted via the input

```
(menu "PHYSICS OF SOUND TUTORIAL"
"Introduction"->st_1_1_1,
"Source"->st_1_2_1,
"Medium"->st_1_3_1,
"Sound"->st_1_4_1,
"Detector"->st_1_5_1,
"Intro Test"->st_1_10_1,
"Overall Test"->st_7_1)
```

FIGURE 20: cfd_menu Example

handler for the canvas. The next state is chosen and executed based on the interpretation of response options for the current state. The sequence for presenting a state is the same: present the actions, allow SunView to accept the response, interpret the response and then go to the next state. At anytime during the execution of the tutorial, the user has the option of exiting the tutorial.

1. Present Actions

The process of presenting the actions of a state is first to determine the region that the action is to occur. Once the region is determined, the action is then executed within that region as described previously. If the action list consists of more than one action, each action in the action list is executed.

2. Interpret Response

The response is first interpreted by Sunview event handling procedures to determine exactly what the response was. Then, the corresponding response is found in the response list. If the next state to go to relies on past answer history, the assert list is included in the response evaluation. Once the appropriate response and past history have been found in the response list, the next state is executed. If no match with the response list is identified, the software remains in the same state and waits for further input. However, most states will

specify a default transition to be taken in such a case, i. e., users will not be left "stranded" at some point in the tutorial.

V. CONCLUSION AND FUTURE RESEARCH

A. SUMMARY OF CONTRIBUTIONS

A Concept-Flow Diagram (CFD) is a representation of the information presented in a computer-aided tutorial. The CFD highlights the concepts that are prerequisites to the presentation of other concepts and identifies where verification of mastery is to be performed within the tutorial. A visualization of this information, as provided with the CFD, has two key benefits.

First, the CFD provides a functional basis for the design of tutorials. The designers of the computer-aided tutorial are provided with a tool that assists in the design of the presentation of the tutorial. The CFD allows for a hierarchical presentation of the tutorial, forward and backward presentation-flow, and answer analysis that is history sensitive. Designers are also presented with a tool whose symbology is somewhat familiar and therefore immediately implementable. By using the CFD design methodology, the designers are able to see the presentation flow prior to any programming or coding.

Second, the CFD provides a basis for evaluation of tutorial presentation. By using the CFD methodology, the designers are forced to take a closer look at the presentation flow of the concept goals. The CFD can identify areas where the presentation is presenting too much information. It can also help to identify where the ideal place within a presentation occurs to test the user for comprehension. The CFD strongly encourages non-linear dependency rather than an explicit ordering to the concept-flow of the tutorial presentation.

By highlighting the flow of presentation as a basis for software design, the CFD enables more realistic engineering of computer-based tutorials. Automated strategies, such as the CFD interpreter described in this thesis, are supported by this design technique. In summary, the introduction of the CFD shifts the emphasis of computer-aided instruction design from modelling of student behavior to design of presentation and knowledge verification.

B. RECOMMENDATIONS FOR USE

Concept-Flow Diagrams allow the design of tutorials to take place in a specific context of assumed and demonstrated knowledge. The CFD methodology is immediately useful in tutorials for students with heterogeneous backgrounds. The CFD does not rely on the users to have some common knowledge base in

order to use the tutorial. Therefore, the tutorial can be directed at a more generic audience, from the novice to the expert, without over-challenging or boring the user. The use of this methodology would greatly enhance the tutorials used in organizations, such as the U. S. Navy or the Department of Defense, where the personnel who attend their schools range from the recent high school graduate to individuals with advanced degrees and/ or experience.

Concept-Flow Diagrams are also useful in providing information on how to use interactive software with complex user interfaces. The users of on-line help within a software package, such as an editor or word processor, are usually presented with detailed information about the item requested. The use of Concept-Flow Diagrams would allow the user the option of a detailed explanation or a quick review, pleasing both the novice user and the expert user of the system who may have just forgotten how a function operates. The CFD also can place the help provided in the context of recent commands through the use of the answer analysis ***assert*** and ***past*** non-terminals.

C. RECOMMENDATIONS FOR FUTURE RESEARCH

This thesis concentrated on the design methodology of the presentation flow and representation in a CFD. An extension of this discussion led to answer analysis. A more concentrated effort in the area of answer analysis is well-deserved. This includes more formally describing mastery test flow and usage.

In order to build tools to generate and evaluate Concept-Flow Diagrams, formal semantics and code generation are needed. Therefore, two areas recommended for future research are establishing the formal semantics of Concept-Flow Diagrams and building tools to generate code from Concept-Flow Diagrams.

This leads to actually building the tools to generate and evaluate Concept-Flow Diagrams. The tool or tools could verify such things as too many bubbles per level, points of entry exist via both the MT and concept presentation, and that flow is possible, both forward and backward, from MT to MT or bubble to bubble.

In order to enhance the tutorial presentation, the CFD language can be extended to provide vector graphics capability and animation capability. Currently, the tutorial designers are restricted to stationary icons. The extended capabilities would allow for using moving objects in a task or presentation.

The last recommendation is the transformation from prototype to working interpreter. The interpreter designed is a partially functional prototype. Completion of the prototype and implementation as a production model are necessary.

APPENDIX A

LIST OF CONCEPT GOALS FOR

THE PHYSICS OF SOUND TUTORIAL

1. Introduction and Basic Definitions
   a. Buttonology
   b. Source
   c. Medium
   d. Sound and Ray Path
   e. Detector

2. Sound Characteristics
   a. Frequency
   b. Hertz
   c. Period
   d. Compression
   e. Rarefaction
   f. Longitudinal wave
   g. Amplitude
   h. Effective Pressure Amplitude
   i. Wavelength
   j. Wavefront
   k. Absolute Sound Pressure Level (ABS SPL)
   l. Decibel
   m. Broadband
   n. Tonals

3. Ray Path Transmission and Loss
   a. Direct Path
   b. Reflected Path
   c. Refracted Path
   d. Refracted Surface Reflected Path (RSR)
   e. Limiting Ray
   f. Shadow Zones
   g. Attenuation
   h. Scattering
   i. Spreading/Divergence
   j. Spherical Spreading
   k. Cylindrical Spreading
   l. Multipath Propagation
   m. Critical Angle

4. Ocean Characteristics
   a. Bathymetry
   b. Gradients
   c. Isothermal
   d. Gradient
   e. Thermocline
   f. Sound Channel
   g. Deep Sound Channel (DSC)
   h. Surface Ducts
   i. Bottom Bounce
   j. Mixed Layer
   k. Convergence Zones
   l. Reliable Acoustic Path
   m. Noise
   n. Hydrodynamic Noise
   o. Biological Noise
   p. Seismic Noise
   q. Ocean Traffic Noise
   r. Sea Surface Noise

5. Passive Sonar Equation
   a. Transmission Loss (TL)
   b. Source Level (SL)
   c. Noise Level (NL) - ambient noise, platform noise
   d. Array Gain (AG)
   e. Recognition Differential (RD)
   f. Figure of Merit (FOM)
   g. Signal Excess (SE)

6. Sound Velocity Profile (SVP)
   a. Temperature
   b. Pressure
   c. Salinity
   d. Deep Sound Channel Axis
   e. Noise Spectrum Level
   f. Bandwidth
   g. Doppler
   h. Echo Level

APPENDIX B

PHYSICS OF SOUND TUTORIAL SCRIPT

```
/* AS OF 17 JAN 92, 1430                                    */

(menu "PHYSICS OF SOUND TUTORIAL"
 "Introduction" -> st_1_1_1,
 "Source" -> st_1_2_1,
 "Medium" -> st_1_3_1,
 "Sound" -> st_1_4_1,
 "Detector" -> st_1_5_1,
 "Intro Test" -> st_1_10_1,
 "Overall Test" -> st_7_1)

detector := "detector_sym.icon"

mouse_sym := "mouse_sym.icon"

path := "path_sym.icon"

post := "post_sym.icon"

source := "source_sym.icon"

/********************** BUTTONOLOGY 1.1 ************************/

/********************** INTRODUCTION 1.1.1 **********************/

(st_1_1_1, ( (0, clear), (1, draw, post@(mouseX,mouseY)),
   (2, write, "Welcome to the Physics of Sound tutorial"),
   (5, write, "Upon completion of this tutorial, you will have enough of an understanding
to complete the Physics of Sound module of your OQS"),
   (8, write, "Let's begin . . ."), (0, pause, 15) ),
   st_1_1_5 )
```

```
/************************ MOUSE BUTTONS 1.1.2 ******************/

(st_1_1_5, ( (0, clear),
   (1, draw, mouse_sym@(mouseX, mouseY)),
   (3, write, "In order for you to interact with the computer, you must be familiar with the
keyboard and the mouse. It is assumed that you are already somewhat familiar with the
keyboard since you logged onto the system"),
   (4, write, "Notice the figure in the upper lefthand corner of the screen. This is called the
MOUSE. There should be one hooked up to your terminal."),
   (5, write, "There are 3 buttons on the top of the MOUSE, each one performing a different
function."), (0, pause, 10) ),
   st_1_1_10 )

(st_1_1_10, ( (3, clear),
   (4, clear),
   (5, clear),
   (3, write, "The LEFT button performs an ACTION when clicked once."),
   (7, write, "Click the LEFT button on the MOUSE once when you are ready to continue."),
   (0, input, mouse),  /* allow input with mouse */
   (9, input, keyboard) ),  /* allow input with keyboard only in */
                  /* region 9                    */
   ( (click-left, (assert, left), st_1_1_15),
     (click-middle & (past st_1_1_40 2X_wrong), st_1_1_45),
     (click-middle & (past st_1_1_35 wrong_ans), st_1_1_40),
     (click-middle, (assert, mid), st_1_1_35),
     (click-right & (past st_1_1_40 2X_wrong), st_1_1_45),
     (click-right & (past st_1_1_35 wrong_ans), st_1_1_40),
     (click-right, (assert, right), st_1_1_35),
     (15 seconds, (assert, wait), st_1_1_50),
     (("Help" | "I do not know") & (past st_1_1_10 help), st_1_1_45),
     ("Help" | "I do not know", (assert, help), st_1_1_65),
     ((past st_1_1_10 wrong_ans) & (past st_1_1_35 wrong_ans), st_1_1_45),
     (assert, wrong_ans), st_1_1_35 ) )

(st_1_1_15, ( (3, clear), (7, clear), (9, clear),
   (5, write, "Very good. That is the correct button.") ),
   st_1_1_20)
```

```
(st_1_1_20, ( (5, clear),
   (3, write, "At this time, the MIDDLE and RIGHT mouse buttons have no function.
Therefore, you will not have to use them."),
   (7, write, "Click the LEFT mouse button when you are ready to continue."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (click-left, (assert, left), st_1_1_25),
    (click-middle & (past st_1_1_40 2X_wrong), st_1_1_45),
    (click-middle & (past st_1_1_35 wrong_ans), st_1_1_40),
    (click-middle, (assert, mid), st_1_1_35),
    (click-right & (past st_1_1_40 2X_wrong), st_1_1_45),
    (click-right & (past st_1_1_35 wrong_ans), st_1_1_40),
    (click-right, (assert, right), st_1_1_35),
    (15 seconds, (assert, wait), st_1_1_80),
    (("Help" | "I do not know") & (past st_1_1_20 help), st_1_1_45),
    ("Help" | "I do not know", (assert, help), st_1_1_40),
    ((past st_1_1_20 wrong_ans) & (past st_1_1_35 wrong_ans), st_1_1_45),
    (assert, wrong_ans), st_1_1_35) )

(st_1_1_25, ( (3, clear), (7, clear),
   (3, write, "The position of the cursor on the screen can be changed by moving the
MOUSE on the pad."),
   (4, write, "Try moving the cursor to different positions on the screen using the mouse."),
   (7, write, "Once you are comfortable with moving the cursor, position the cursor over the
CONTINUE button at the top of the screen and click the left mouse button once."),
   (0, input, mouse),
   (9, input, keyboard)  ),
   ( (mouse-move, st_1_1_30),
    (click-continue & (past st_1_5_30 wrong_ans), st_1_5_1),
    (click-continue & (past st_1_2_30 wrong_ans), st_1_2_5),
    (click-continue & ((past st_1_6_25 help) | (past st_1_6_25 wrong_ans)),
              (assert, continue), st_1_6_1),
    (click-continue & ((past st_1_6_5 help) | (past st_1_6_5 wrong_ans)),
              (assert, continue), st_1_6_1),
    (click-continue & ((past st_1_6_1 help) | (past st_1_6_1 wrong_ans)),
              (assert, continue), st_1_6_1),
    (click-continue, (assert, continue), st_1_1_110),
    (15 seconds, (assert, wait),  st_1_1_70),
    ("Help" | "I do not know", (assert, help), st_1_1_75),
    (assert, wrong_ans), st_1_1_75))
```

```
(st_1_1_30, ( (3, clear), (4, clear), (7, clear), (9,clear),
   (4, write, "Good. Try moving the position of the cursor using the mouse again."),
   (7, write, "When you are comfortable with moving the cursor, move the cursor to the
CONTINUE button at the top of the screen and click the LEFT mouse button once."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (mouse-move, st_1_1_30),
    (click-continue, (assert, continue), st_1_1_110),
    (5 seconds, (assert, wait), st_1_1_85),
    ("Help" | "I do not know" | "How", (assert, help), st_1_1_75),
    (assert, wrong_ans), st_1_1_75) )

(st_1_1_35, ( (3, clear), (5, clear), (9,clear),
   (3, write, "That is not the correct button. Try again.") ),
   ( ((past st_1_1_10 mid | past st_1_1_10 right | past st_1_1_10 wrong_ans),
     (assert, wrong_ans), st_1_1_10),
     ((past st_1_1_20 mid | past st_1_1_20 right | past st_1_1_20 wrong_ans),
     (assert, wrong_ans), st_1_1_20),
     ((past st_1_1_20 wrong_ans), (assert, wrong_ans), st_1_1_20) ) )

(st_1_1_40, ( (3, clear), (5, clear), (9, clear),
   (3, write, "Remember, the MIDDLE and RIGHT buttons on the mouse DO NOT perform
any function. Therefore, click the LEFT button on the mouse once to perform an
ACTION."),
   (7, write, "Try the exerise again.") ),
   ( ((past st_1_1_10 mid | past st_1_1_10 right) & (past st_1_1_35 wrong_ans),
     (assert, 2X_wrong), st_1_1_10),
     ((past st_1_1_20 mid | past st_1_1_20 right) & (past st_1_1_35 wrong_ans),
     (assert, 2X_wrong), st_1_1_20),
     ((past st_1_1_20 help), st_1_1_20) ) )

(st_1_1_45, ( (3, clear), (5, clear),
   (3, write, "You seem to be having trouble."),
   (4, write, "Before progressing any further, get assistance from your instructor."), (0, quit)
), st_1_1_45)
```

```
(st_1_1_50, ( (7, write, "Don't be afraid to try pressing a button on the mouse."),
   (0, input, mouse),
   (9, input, keyboard)  ),
   ( (15 seconds & (past st_1_1_50 wait_wait), st_1_1_55),
    (15 seconds & (past st_1_1_10 wait), (assert, wait_wait), st_1_1_50),
    (click-left, (assert, left), st_1_1_15),
    (click-middle & (past st_1_1_40 2X_wrong), st_1_1_45),
    (click-middle & (past st_1_1_35 wrong_ans), st_1_1_40),
    (click-middle, (assert, mid), st_1_1_35),
    (click-right & (past st_1_1_40 2X_wrong), st_1_1_45),
    (click-right & (past st_1_1_35 wrong_ans), st_1_1_40),
    (click-right, (assert, right), st_1_1_35),
    (("Help" | "I do not know") & (past st_1_1_10 help), st_1_1_45),
    ("Help" | "I do not know", (assert, help), st_1_1_65),
    ((past st_1_1_10 wrong_ans & past st_1_1_35 wrong_ans), st_1_1_45),
    (assert, wrong_ans), st_1_1_35 ) )

(st_1_1_55, ( (0, clear),
   (5, write, "You have not taken action."),
   (8, write, "Do you want help?"),
   (9, input, keyboard) ),
   (("Yes" | "YES" | "yes" | "Y" | "y", (assert, help), st_1_1_45),
    ("No" | "NO" | "no" | "N" | "n", (assert, check_pt), st_1_1_60),
    (assert, wrong_ans), st_1_1_45) )

(st_1_1_60, ( (0, clear),
   (5, write, "It is assumed that you wish to EXIT."), (0,quit) ),
   st_1_1_60 )

(st_1_1_65, ( (3, clear), (5, clear),
   (3, write, "Let's take a look at the MOUSE again.") ),
   (assert, help), st_1_1_10)
```

```
(st_1_1_70, ( (7, write, "Don't be afraid to try pressing a button on the mouse."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (15 seconds & (past st_1_1_70 wait_wait), st_1_1_55),
    (15 seconds & (past st_1_1_10 wait), (assert, wait_wait), st_1_1_70),
    (click-left & (past st_1_1_140 help | past st_1_1_140 wrong_ans), st_1_1_140),
    (click-left, (assert, left), st_1_1_15),
    (click-middle & (past st_1_1_40 2X_wrong), st_1_1_45),
    (click-middle & (past st_1_1_35 wrong_ans), st_1_1_40),
    (click-middle, (assert, mid), st_1_1_35),
    (click-right & (past st_1_1_40 2X_wrong), st_1_1_45),
    (click-right & (past st_1_1_35 wrong_ans), st_1_1_40),
    (click-right, (assert, right), st_1_1_35),
    (("Help" | "I do not know") & (past st_1_1_10 help), st_1_1_45),
    ("Help" | "I do not know", (assert, help), st_1_1_65),
    ((past st_1_1_10 wrong_ans & past st_1_1_35 wrong_ans), st_1_1_45),
    (assert, wrong_ans), st_1_1_35 ) )
```

```
(st_1_1_75, ( (3, clear), (7, clear), (9, clear),
   (3, write, "You did not find the CONTINUE button. You must first move the position
of the cursor so that the cursor is positioned within the rectangle labelled 'CONTINUE.'
Once you have positioned the cursor, click the left mouse button once."),
   (5, write, "Locate the CONTINUE button and click the left mouse button once."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (click-continue & (past st_1_1_175 help | past st_1_1_175 wrong_ans),
    st_1_1_160),
   (click-continue & (past st_1_1_165 help | past st_1_1_165 wrong_ans),
    st_1_1_160),
   (click-continue & (past st_1_1_160 help | past st_1_1_160 wrong_ans),
    st_1_1_160),
   (click-continue & (past st_1_1_150 help | past st_1_1_150 wrong_ans),
    st_1_1_150),
   (click-continue & (past st_1_1_145 help | past st_1_1_145 wrong_ans),
    st_1_1_140),
   (click-continue & (past st_1_1_140 help | past st_1_1_140 wrong_ans),
    st_1_1_140),
   (click-continue & (past st_1_1_116 help | past st_1_1_116 wrong_ans),
    st_1_1_110),
   (click-continue & (past st_1_1_115 help | past st_1_1_115 wrong_ans),
    st_1_1_110),
   (click-continue & (past st_1_1_110 help | past st_1_1_110 wrong_ans),
    st_1_1_110),
   (click-continue, (assert, continue), st_1_6_1),
   (5 seconds, (assert, wait), st_1_1_90),
   ("Help" | "I do not know", (assert, help), st_1_1_45),
   (assert, wrong_ans), st_1_1_45) )
```

```
(st_1_1_80, ( (7, write, "Don't be afraid to try pressing a button on the mouse."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (15 seconds & (past st_1_1_80 wait_wait), st_1_1_55),
    (15 seconds & (past st_1_1_20 wait), (assert, wait_wait), st_1_1_80),
    (click-left, (assert, left), st_1_1_25),
    (click-middle & (past st_1_1_40 2X_wrong), st_1_1_45),
    (click-middle & (past st_1_1_35 wrong_ans), st_1_1_40),
    (click-middle, (assert, mid), st_1_1_35),
    (click-right & (past st_1_1_40 2X_wrong), st_1_1_45),
    (click-right & (past st_1_1_35 wrong_ans), st_1_1_40),
    (click-right, (assert, right), st_1_1_35),
    (("Help" | "I do not know") & (past st_1_1_10 help), st_1_1_45),
    ("Help" | "I do not know", (assert, help), st_1_1_65),
    ((past st_1_1_10 wrong_ans & past st_1_1_35 wrong_ans), st_1_1_45),
    (assert, wrong_ans), st_1_1_35 ) )

(st_1_1_85, ( (7, write, "Don't be afraid to try pressing a button on the mouse."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (15 seconds & (past st_1_1_85 wait_wait), st_1_1_55),
    (15 seconds & (past st_1_1_20 wait), (assert, wait_wait), st_1_1_85),
    (mouse-move, st_1_1_30),
    (click-continue, (assert, continue), st_1_6_1),
    ("Help" | "I do not know" | "How", (assert, help), st_1_1_75),
    (assert, wrong_ans), st_1_1_75 ) )

(st_1_1_90, ( (7, write, "Don't be afraid to try pressing a button on the mouse."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (15 seconds & (past st_1_1_90 wait_wait), st_1_1_55),
    (15 seconds & (past st_1_1_20 wait), (assert, wait_wait), st_1_1_90),
    (click-continue, (assert, continue), st_1_6_1),
    ("Help" | "I do not know", (assert, help), st_1_1_45),
    (assert, wrong_ans), st_1_1_45 ) )

/************************ End MOUSE BUTTONS ************************/
```

```
/********************** MAIN FRAME BUTTONS **********************/

(st_1_1_110, ( (0, clear),
   (1, write, "Notice the 3 buttons that appear at the top of the screen."),
   (2, write, "The buttons are labelled HELP, CONTINUE, and EXIT."),
   (4, write, "These 3 buttons will always appear at the top of the screen while using the
Physics of Sound tutorial."),
   (7, write, "Let's take a look at how these buttons operate."),
   (9, write, "Click the mouse on the CONTINUE button  when you are ready to move on
. . ."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (click-continue, (assert, continue), st_1_1_120),
    (5 seconds, (assert, wait), st_1_1_115),
    ("Help" | "I do not know", (assert, help), st_1_1_75),
    (assert, wrong_ans), st_1_1_75 ) )

(st_1_1_115, ( (7, write, "Don't be afraid to try pressing a button on the mouse."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (15 seconds & (past st_1_1_115 wait_wait), st_1_1_116),
    (15 seconds & (past st_1_1_110 wait), (assert, wait_wait), st_1_1_115),
    (click-continue, (assert, continue), st_1_1_120),
    ("Help" | "I do not know", (assert, help), st_1_1_75),
    (assert, wrong_ans), st_1_1_75 ) )

(st_1_1_116, ( (0, clear),
   (5, write, "You have not taken action."),
   (8, write, "Do you want help?"),
   (9, input, keyboard) ),
   (("Yes" | "YES" | "yes" | "Y" | "y", (assert, help), st_1_1_75),
    ("No" | "NO" | "no" | "N" | "n", (assert, check_pt), st_1_1_60),
    (assert, wrong_ans), st_1_1_75 ) )
```

```
/************************** HELP BUTTON **********************/

(st_1_1_120, ( (0, clear),
  (1, write, "The HELP button can be used at any time. The button is designed to assist you
when you are unable to determine the function of a specific feature on the screen."),
  (4, write, "To use the HELP button, move the cursor to the button labelled HELP located
in the upper lefthand corner of the screen and click the left mouse button once."),
  (5, write, "You will then be guided through the use of the particular function that you
requested."),
  (7, write, "Click the mouse on the HELP button"),
  (0, input, mouse),
  (9, input, keyboard) ),
  ( (click-help & (past st_1_6_30 help | past st_1_1_30 wrong_ans), (assert, help_button),
st_1_6_20),
    (click-help & (past st_1_6_25 help | past st_1_1_25 wrong_ans), (assert, help_button),
st_1_6_20),
    (click-help & (past st_1_6_20 help | past st_1_1_20 wrong_ans), (assert, help_button),
st_1_6_20),
    (click-help, (assert, help_button), st_1_1_140),
    (5 seconds, (assert, wait), st_1_1_125),
    ("Help" | "I do not know", (assert, help), st_1_1_130),
    (assert, wrong_ans), st_1_1_130) )

(st_1_1_125, ( (7, write, "Don't be afraid to try pressing a button on the mouse."),
  (0, input, mouse),
  (9, input, keyboard) ),
  ( (15 seconds & (past st_1_1_130 wait & past st_1_1_125 wait_wait), st_1_1_45),
    (15 seconds & (past st_1_1_130 wait), (assert, wait_wait), st_1_1_125),
    (15 seconds & (past st_1_1_120 wait & past st_1_1_125 wait_wait), st_1_1_135),
    (15 seconds & (past st_1_1_120 wait), (assert, wait_wait), st_1_1_125),
    (click-help, (assert, help_button), st_1_1_140),
    ("Help" | "I do not know", (assert, help), st_1_1_130),
    (assert, wrong_ans), st_1_1_130 ) )
```

```
(st_1_1_130, ((1, clear), (4, clear), (5, clear), (7, clear),
   (1, write, "You did not correctly identify the HELP button. First, move the position of
the cursor so that the cursor is positioned within the rectangle labelled 'HELP.' Once you
have positioned the cursor, click the left mouse button once."),
   (5, write, "Locate the HELP button and click the left mouse button once."),
   (0, input, mouse),
   (9, input, mouse) ),
   ( (click-help, (assert, help_button), st_1_1_140),
    (5 seconds, (assert, wait), st_1_1_125),
    ("Help" | "I do not know", (assert, help), st_1_1_45),
    (assert, wrong_ans), st_1_1_45 ) )

(st_1_1_135, ( (0, clear),
   (5, write, "You have not taken action."),
   (8, write, "Do you want help?"),
   (9, input, keyboard) ),
   (("Yes" | "YES" | "yes" | "Y" | "y", (assert, help), st_1_1_130),
    ("No" | "NO" | "no" | "N" | "n", (assert, check_pt), st_1_1_60),
    (assert, wrong_ans), st_1_1_130 ) )

/********************** End HELP BUTTON ***************************/
```

```
/********************** CONTINUE BUTTON ***************************/

(st_1_1_140, ( (0, clear),
   (1, write, "You have already seen the use of the CONTINUE button. The CONTINUE
button is used to perform an action, such as moving the position of an icon or to accept
input."),
   (4, write, "To use the CONTINUE button, move the cursor to the button labelled
CONTINUE located in the upper left of the screen and click the left mouse button once."),
   (5, write, "Move the cursor to the CONTINUE button and click left mouse button once
when you are ready to continue . . ."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (click-continue & (past st_1_5_10 wrong_ans), st_1_5_1),
    (click-continue & (past st_1_4_5 wrong_ans), st_1_4_5),
    (click-continue & (past st_1_4_1 wrong_ans), st_1_4_1),
    (click-continue & (past st_1_3_5 wrong_ans), st_1_3_5),
    (click-continue & (past st_1_3_1 wrong_ans), st_1_3_1),
    (click-continue & (past st_1_3_1 wrong_ans), st_1_3_1),
    (click-continue, (assert, continue), st_1_1_160),
    (5 seconds, (assert, wait), st_1_1_145),
    ("Help" | "I do not know", (assert, help), st_1_1_75),
    (assert, wrong_ans), st_1_1_75 ) )



(st_1_1_145, ( (7, write, "Don't be afraid to try pressing a button on the mouse."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (15 seconds & (past st_1_1_140 wait & past st_1_1_145 wait_wait), st_1_1_150),
    (15 seconds & (past st_1_1_140 wait), (assert, wait_wait), st_1_1_145),
    (click-continue, (assert, continue), st_1_1_160),
    ("Help" | "I do not know", (assert, help), st_1_1_75),
    (assert, wrong_ans), st_1_1_75 ) )

(st_1_1_150, ( (0, clear),
   (5, write, "You have not taken action."),
   (8, write, "Do you want help?"),
   (9, input, keyboard) ),
   (("Yes" | "YES" | "yes" | "Y" | "y", (assert, help), st_1_1_75),
    ("No" | "NO" | "no" | "N" | "n", (assert, check_pt), st_1_1_60),
    (assert, wrong_ans), st_1_1_75 ) )

/********************** End CONTINUE BUTTON **********************/
```

```
/************************** EXIT BUTTON **************************/

(st_1_1_160, ( (0, clear),
   (1, write, "The EXIT button is used when you are ready to stop working on the Physics
of Sound tutorial."),
   (3, write, "If you are in the midddle of the tutorial when you EXIT, the system will keep
track of where you left off and you will restart at this point the the next time you logon."),
   (4, write, "The EXIT button will always be located in the upper righthand corner of the
screen."),
   (6, write, "We will not experiment with the EXIT button at this time since it is assumed
you wish to continue on with the tutorial."),
   (7, write, "Click the mouse on the CONTINUE button when you are ready to move on .
. ."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (click-continue, (assert, continue), st_1_6_1),
    (5 seconds, (assert, wait), st_1_1_165),
    ("Help" | "I do not know", (assert, wait), st_1_1_75),
    (assert, wrong_ans), st_1_1_75 ) )

(st_1_1_165, ( (7, write, "Don't be afraid to try pressing a button on the mouse."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (15 seconds & (past st_1_1_160 wait & past st_1_1_165 wait_wait), st_1_1_175),
    (15 seconds & (past st_1_1_160 wait), (assert, wait_wait), st_1_1_165),
    (click-continue, (assert, continue), st_1_6_1),
    ("Help" | "I do not know", (assert, wait), st_1_1_75),
    (assert, wrong_ans), st_1_1_75 ) )

(st_1_1_175, ( (0, clear),
   (5, write, "You have not taken action."),
   (8, write, "Do you want help?"),
   (9, input, keyboard) ),
   (("Yes" | "YES" | "yes" | "Y" | "y", (assert, help), st_1_1_75),
    ("No" | "NO" | "no" | "N" | "n", (assert, check_pt), st_1_1_60),
    (assert, wrong_ans), st_1_1_75 ) )

/************************ End EXIT BUTTON ************************/
```

```
/************************ MASTERY TEST 1.6 ************************/

(st_1_6_1, ( (0, clear),
   (1, draw, mouse_sym),
   (4, write, "As you know, the mouse is required for interaction with the Physics of Sound
tutorial."),
   (7, write, "Click the left mouse button once on the CONTINUE button to move on to the
next task."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (click-continue, (assert, continue), st_1_6_20),
     (5 seconds, (assert, wait), st_1_6_5),
     ("Help" | "I do not know", (assert, help), st_1_1_25),
     (assert, wrong_ans), st_1_1_25 ) )

(st_1_6_5, ( (7, write, "Don't be afraid to try pressing a button on the mouse."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (15 seconds & (past st_1_6_5 wait_wait), st_1_6_10),
     (15 seconds & (past st_1_6_1 wait), (assert, wait_wait), st_1_6_5),
     (click-continue, (assert, continue), st_1_1_110),
     ("Help" | "I do not know", (assert, help), st_1_1_25),
     (assert, wrong_ans), st_1_1_25 ) )

(st_1_6_10, ( (0, clear),
   (5, write, "You have not taken action."),
   (8, write, "Do you want help?"),
   (9, input, keyboard) ),
   (("Yes" | "YES" | "yes" | "Y" | "y", (assert, help), st_1_1_25),
    ("No" | "NO" | "no" | "N" | "n", (assert, check_pt), st_1_1_60),
    (assert, wrong_ans), st_1_1_25 ) )

(st_1_6_20, ( (0, clear),
   (1, write, "Experiment with the HELP button until you feel comfortable with its use."),
   (7, write, "Click the mouse once on the CONTINUE button when you are ready to move
on . . ."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (click-help, (assert, help_button), st_1_6_20),
     (click-continue, (assert, continue), st_1_2_1), /* NEXT MT */
     (5 seconds, (assert, wait), st_1_6_25),
     ("Help" | "I do not know", (assert, help), st_1_1_120),
     (assert, wrong_ans), st_1_1_120 ) )
```

```
(st_1_6_25, ( (7, write, "Don't be afraid to try pressing a button on the mouse."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (15 seconds & (past st_1_6_20 wait_wait), st_1_6_25),
    (15 seconds & (past st_1_6_20 wait), (assert, wait_wait), st_1_6_30),
    (click-help, (assert, help_button), st_1_6_20),
    (click-continue, (assert, continue), st_1_2_1), /* NEXT MT */
    ("Help" | "I do not know", (assert, help), st_1_1_120),
    (assert, wrong_ans), st_1_1_120 ) )

(st_1_6_30, ( (0, clear),
   (5, write, "You have not taken action."),
   (8, write, "Do you want help?"),
   (9, input, keyboard) ),
   (("Yes" | "YES" | "yes" | "Y" | "y", (assert, help), st_1_1_120),
    ("No" | "NO" | "no" | "N" | "n", (assert, check_pt), st_1_1_60),
    (assert, wrong_ans), st_1_1_120 ) )

/*************************** End MT 1.6 ************************/

/*************************** SOURCE 1.2 ************************/

(st_1_2_1, ( (0, clear),
   (1, write, "Now that you have become familiar with the use of the mouse, let's move on
to another topic."),
   (2, write, "In order to study the physics of sound, there are 3 basic properties that must
exist:"),
   (3, write, "SOURCE, SOUND, and DETECTOR"),
   (4, write, "First, let's dicuss the term SOURCE."),
   (0, pause, 30) ),
   st_1_2_5 )
```

(st_1_2_5, ((0, clear),
(1, draw, source),
(3, write, "A SOURCE is any object that moves or vibrates disturbing the medium around it."),
(5, write, "The SOURCE is represented by a FISH, as is shown in the upper lefthand corner of the screen."),
(8, write, "Locate the SOURCE by moving the position of the mouse to the FISH and click the left mouse button once . . ."),
(1, input, mouse)),
((click-left, (assert, left), st_1_2_10),
((click-middle | click-right) & (past st_1_2_25 wrong_ans), (assert, help), st_1_2_30),
(click-middle, (assert, mid), st_1_2_25),
(click-right, (assert, right), st_1_2_25),
(5 seconds, (assert, wait), st_1_2_15),
("Help" | "I do not know", (assert, help), st_1_2_30),
(assert, wrong_ans), st_1_2_30))

(st_1_2_10, ((3, clear), (5, clear), (8, clear),
(3, write, "Great! You were able to locate the SOURCE."),
(4, write, "Now, what can you do with the SOURCE?"),
(5, write, "You cannot move the SOURCE since in a real world situation you will have no control over the location of a source."),
(0, pause, 45)),
st_1_3_1)

(st_1_2_15, ((7, write, "Don't be afraid to move the cursor over the SOURCE location and pressing a button on the mouse."),
(0, input, mouse),
(9, input, keyboard)),
((15 seconds & (past st_1_2_15 wait_wait), st_1_2_20),
(15 seconds & (past st_1_2_5 wait), (assert, wait_wait), st_1_2_15),
(click-left, (assert, left), st_1_2_10),
((click-middle | click-right) & (past st_1_2_25 wrong_ans), (assert, help), st_1_2_30),
(click-middle, (assert, mid), st_1_2_25),
(click-right, (assert, right), st_1_2_25),
("Help" | "I do not know", (assert, help), st_1_2_30),
(assert, wrong_ans), st_1_2_30))

```
(st_1_2_20, ( (0, clear),
   (5, write, "You have not taken action."),
   (8, write, "Do you want help?"),
   (9, input, keyboard) ),
   (("Yes" | "YES" | "yes" | "Y" | "y", (assert, help), st_1_2_30),
    ("No" | "NO" | "no" | "N" | "n", (assert, check_pt), st_1_1_60),
    (assert, wrong_ans), st_1_2_30 ) )

(st_1_2_25, ( (3, clear), (5, clear), (8, clear),
   (3, write, "That is the incorrect button.  Remember, the MIDDLE and RIGHT mouse
buttons do not perform any function. Try again.") ),
   st_1_2_5)

(st_1_2_30, ( (3, clear), (5, clear), (8, clear),
   (3, write, "Let's review how to move the cursor using the mouse.") ),
   (assert, help), st_1_1_25 )

/***************************** END SOURCE ************************/

/************************ MASTERY TEST 1.7 ***********************/

/* DO NOT NEED MT AFTER SOURCE MOD -- TOO SIMPLISTIC            */

/*****************************************************************/

/************************** MEDIUM 1.3 ***************************/

(st_1_3_1, ( (0, clear),
   (1, write, "A MEDIUM is a substance regarded as the means of transmission of a force
or effect."),
   (4, write, "Becasue the POST deals specifically with underwater sound, the MEDIUM
through which the sound travels is SEA WATER."),
   (7, write, "The successful transmission of sound is dependent on the ability of the
MEDIUM to react to changes in pressure originated by the sound SOURCE."),
   (8, write, "Click the CONTINUE button when you are ready to move on . . ."),
   (0, input, mouse) ),
   ( (click-continue, (assert, continue), st_1_3_5),
     (300 seconds, st_1_3_5),
     (assert, wrong_ans), st_1_1_140 ) )
```

```
(st_1_3_5, ( (0, clear),
  (1, write, "To pass on sound, the MEDIUM must have the capability to respond to
variations or changes in the SOURCE pressure fluctuations."),
  (4, write, "Sea water possess the quality called ELASTICITY. This means that the sound
pressures causes physical movement of the water molecules which return to their normal
state following the passage of SOUND."),
  (7, write, "The travel of sound through a MEDIUM is called PROPOGATION."),
  (8, write, "Click the CONTINUE button when you are ready to move on . . ."),
  (0, input, mouse) ),
  ( (click-continue, (assert, continue), st_1_4_1),
   (300 seconds, st_1_4_1),
   (assert, wrong_ans), st_1_1_140 ) )

/************************** End MEDIUM ****************************/

/************************** MASTERY TEST 1.8 ***********************/

/* DO NOT NEED MT AFTER MEDIUM MODULE -- TOO SIMPLISTI       */

/*****************************************************************/

/************************** SOUND 1.4 *****************************/

(st_1_4_1, ( (0, clear),
  (1, draw, path),
  (3, write, "SOUND is a mechanical wave motion that is generated or propagated in an
elastic MEDIUM."),
  (4, write, "SOUND is represented by a line. Since SOUND has direction, the line will
point in the direction that the sound is travelling. In the upper lefthand corner of the screen,
you can see a SOURCE with its SOUND."),
  (8, write, "Click the CONTINUE button when you are ready to move on . . ."),
  (0, input, mouse) ),
  ( (click-continue, (assert, continue), st_1_4_5),
   (300 seconds, st_1_4_5),
   (assert, wrong_ans), st_1_1_140 ) )
```

```
(st_1_4_5, ( (3, clear), (4, clear),
   (3, write, "This line is called a RAY PATH."),
   (8, write, "Click the CONTINUE button when you are ready to move on . . ."),
   (0, input, mouse) ),
   ( (click-continue, (assert, continue), st_1_5_1),
    (300 seconds, st_1_5_1),
    (assert, wrong_ans), st_1_1_140 ) )

/********************** End SOUND **********************************/

/********************** MASTERY TEST 1.9 ***************************/

/* DO NOT NEED MT AFTER SOUND MOD -- TOO SIMPLISTIC                 */

/*********************************************************************/

/*************************** DETECTOR 1.5 ***************************/

(st_1_5_1, ( (0, clear),
   (1, draw, detector),
   (2, write, "A DETECTOR is the RECEIVER of SOUND."),
   (4, write, "The DETECTOR is represented by a _____ as is located in the
upper lefthand corner of the screen."),
   (5, write, "Locate the DETECTOR by placing the cursor in the DETECTOR and click
the left mouse button once."),
   (1, input, mouse) ),
   ( (click-left, (assert, left), st_1_5_10),
    ((click-middle | click-right) & (past st_1_5_25 wrong_ans), (assert, help), st_1_2_30),
    (click-middle, (assert, mid), st_1_5_25),
    (click-right, (assert, right), st_1_5_25),
    (5 seconds, (assert, wait), st_1_5_15),
    ("Help" | "I do not know", (assert, help), st_1_5_30),
    (assert, wrong_ans), st_1_5_30 ) )
```

```
(st_1_5_10, ( (3, clear), (5, clear), (8, clear),
   (3, write, "Great! You were able to locate the DETECTOR."),
   (4, write, "Now, what can you do with the DETECTOR?"),
   (7, write, "The position of the DETECTOR can be determined by you. To change the
position of the DETECTOR, position the cursor inside the DETECTOR. Hold down the
left mouse button and drag the mouse until the DETECTOR moves to the position you want
it and then release the left mouse button."),
   (8, write, "Click the CONTINUE button when you are ready to move on . . ."),
   (0, input, mouse) ),
   ( (click-continue, (assert, continue), st_1_5_35),
    (300 seconds, st_1_5_35),
    (assert, wrong_ans), st_1_1_140 ) )

(st_1_5_15, ( (7, write, "Don't be afraid to move the cursor over the DETECTOR location
and pressing a button on the mouse."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (15 seconds & (past st_1_5_15 wait_wait), st_1_5_20),
    (15 seconds & (past st_1_5_5 wait), (assert, wait_wait), st_1_5_15),
    (click-left, (assert, left), st_1_5_10),
    ((click-middle | click-right) & (past st_1_5_25 wrong_ans), (assert, help), st_1_5_30),
    (click-middle, (assert, mid), st_1_5_25),
    (click-right, (assert, right), st_1_5_25),
    ("Help" | "I do not know", (assert, help), st_1_5_30),
    (assert, wrong_ans), st_1_5_30 ) )

(st_1_5_20, ( (0, clear),
   (5, write, "You have not taken action."),
   (8, write, "Do you want help?"),
   (9, input, keyboard) ),
   (("Yes" | "YES" | "yes" | "Y" | "y", (assert, help), st_1_5_30),
    ("No" | "NO" | "no" | "N" | "n", (assert, check_pt), st_1_1_60),
    (assert, wrong_ans), st_1_5_30 ) )

(st_1_5_25, ( (3, clear), (5, clear), (8, clear),
   (3, write, "That is the incorrect button. Remember, the MIDDLE and RIGHT mouse
buttons do not perform any function. Try again.") ),
   (assert, wrong_ans), st_1_5_1 )

(st_1_2_30, ( (3, clear), (5, clear), (8, clear),
   (3, write, "Let's review how to move the cursor using the mouse.") ),
   (assert, help), st_1_1_25 )
```

```
(st_1_5_35, ( (2, clear), (3, clear), (4, clear), (7, clear),
   (2, write, "Try changing the position of the DETECTOR."),
   (4, write, "Click the CONTINUE button when you are ready to move on . . ."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (click-left, (assert, move_detector), st_1_5_50),
    (click-continue, (assert, continue), st_1_10_1),
    (15 seconds, (assert, wait), st_1_5_45),
    ("Help" | "I do not know", (assert, help), st_1_5_1),
    (assert, wrong_ans), st_1_5_1 ) )

(st_1_5_40, ( (0, clear, detector@(detector.x, detector.y)),
   (1, draw, detector@(mouseX, mouseY) ) ),
   st_1_5_35)

(st_1_5_45, ( (7, write, "Don't be afraid to change the DETECTOR location."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (15 seconds & (past st_1_5_45 wait_wait), st_1_5_20),
    (15 seconds & (past st_1_5_35 wait), (assert, wait_wait), st_1_5_45),
    (click-continue, (assert, continue), st_1_10_1),
    ("Help" | "I do not know", (assert, help), st_1_5_1),
    (assert, wrong_ans), st_1_5_1 ) )

(st_1_5_50, ( (1, clear, detector@(detector.x, detector.y)),
   (1, draw, detector@(mouseX, mouseY)) ),
   st_1_5_35)

/************************** End DETECTOR **************************/

/************************** MASTERY TEST 1.10 **********************/

(st_1_10_1, ( (0, clear),
   (1, draw, source@(mouseX, mouseY) ),
   (1, draw, path@(mouseX, mouseY) ),
   (1, draw, detector@(mouseX, mouseY) ),
   (7, write, "Move the DETECTOR so that the SOUND will hit the DETECTOR."),
   (0, input, mouse) ),
   ( (click-left, (assert, move_detector), st_1_10_10),
    (45 seconds, (assert, wait), st_1_10_20),
    st_1_10_20) )
```

```
(st_1_10_10, ( (1, clear, detector@(detector.x, detector.y)),
   (1, draw, detector@(mouseX, mouseY)) ),
   ( (click-left & (mouseX <= sound.x + halfwid & mouseX >= sound.x - halfwid &
mouseY <= sound.y + halfht & mouseY >= sound.y - halfht), (assert, hit), st_1_10_15),
     (click-left & ( (past st_1_10_10 miss_left) |
                                    (past st_1_10_10 miss_right) |
                                    (past st_1_10_10 miss_low) |
                                    (past st_1_10_10 miss_high)), (assert, help),
st_1_10_30),
     (click-left & (mouseX < sound.x - halfwid), (assert, miss_left), st_1_10_20),
     (click-left & (mouseX > sound.x + halfwid), (assert, miss_right), st_1_10_20),
     (click-left & (mouseY < sound.y - halfht), (assert, miss_low), st_1_10_20),
     (click-left & (mouseY > sound.y + halfht), (assert, miss_high), st_1_10_20),
     (45 seconds, (assert, wait), st_1_10_20) ) )

(st_1_10_15, ( (7, clear),
   (7, write, "Very good.  You positioned the DETECTOR correctly.")
),
   st_7_1 )

(st_1_10_20, ( (7, clear),
   (4, write, "Do you need HELP in how to move the DETECTOR?"),
   (9, input, keyboard) ),
   ( (("YES" | "Yes" | "yes" | "y" | "Y"), (assert, help), st_1_5_1),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_10_10 miss_left) &
     ((past st_1_10_30 move_detector) | (past st_1_10_30 miss_again)),
                (assert, help), st_1_1_45),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_10_10 miss_left), st_1_10_25),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_10_10 miss_right) &
                ((past st_1_10_30 move_detector) | (past st_1_10_30 miss_again)),
                (assert, help), st_1_1_45),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_10_10 miss_right), st_1_10_35),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_10_10 miss_low) &
     ((past st_1_10_30 move_detector) | (past st_1_10_30 miss_again)),
      (assert, help), st_1_1_45),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_10_10 miss_low), st_1_10_40),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_10_10 miss_high) &
                ((past st_1_10_30 move_detector) | (past st_1_10_30 miss_again)),
                 (assert, help), st_1_1_45),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_10_10 miss_high), st_1_10_45),
    (300 seconds, (assert, wait), st_1_1_60) ) )
```

```
(st_1_10_25, ( (7, clear),
   (4, write, "You moved the DETECTOR too far to the left."),
   (7, write, "Try again.") ),
   st_1_10_1 )

(st_1_10_30, ( (4, clear), (7, clear),
   (4, write, "To position the DETECTOR correctly, the RAY PATH must hit some part of
the DETECTOR."),
   (7, write, "Move the DETECTOR so that the SOUND will hit the DETECTOR."),
   (1, input, mouse) ),
   ( (click-left, (assert, move_detector), st_1_10_10),
    (assert, miss_again), st_1_10_20 ) )

(st_1_10_35, ( (7, clear),
   (4, write, "You moved the DETECTOR too far to the right."),
   (7, write, "Try again.") ),
   st_1_10_1 )

(st_1_10_40, ( (7, clear),
   (4, write, "You moved the DETECTOR too far to the low."),
   (7, write, "Try again.") ),
   st_1_10_1 )


(st_1_10_45, ( (7, clear),
   (4, write, "You moved the DETECTOR too far to the high."),
   (7, write, "Try again.") ),
   st_1_10_1 )

/************************** End DETECTOR ************************/

/******************** End of INTRO & BASIC DEFS bubble **************/
```

```
/************************ MASTERY TEST 7 ***************************/
/************* This is the overall MT, Level 1 of CFD ***********************/

/* TASK - Direct Path */

(st_7_1, ( (0, clear),
   (1, draw, source@(mouseX, mouseY)), /* center pt of mouse*/
   (1, draw, path@(mouseX, mouseY)), /* Direct Path */
   (1, draw, detector@(mouseX, mouseY)),
   (7, write, "Move the DETECTOR so that the SOUND will hit the DETECTOR."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (click-left, (assert, move_detector), st_1_7_10),
    (45 seconds, (assert, wait), st_1_10_20),
    ("Help" | "I do not know" | "How", (assert, help), st_1_5_10),
    st_1_7_20) )

(st_1_7_10, ( (1, clear, detector@(detector.x, detector.y)),
   (1, draw, detector@(mouseX, mouseY)) ),
   ( (click-left & (mouseX <= sound_direct.x + halfwid &
     mouseX >= sound_direct.x - halfwid &
     mouseY <= sound_direct.y + halfht &
     mouseY >= sound_direct.y - halfht), (assert, hit), st_1_7_15),
    (click-left & ((past st_1_7_10 miss_left) | (past st_1_7_10 miss_right)
                | (past st_1_7_10 miss_low) | (past st_1_7_10 miss_high)),
                 (assert, help), st_1_7_30),
    (click-left & (mouseX < sound_direct.x - halfwid),
     (assert, miss_left), st_1_7_20),
    (click-left & (mouseX > sound_direct.x + halfwid),
     (assert, miss_right), st_1_7_20),
    (click-left & (mouseY < sound_direct.y - halfht),
     (assert, miss_low), st_1_7_20),
    (click-left & (mouseY > sound_direct.y + halfht),
     (assert, miss_high), st_1_7_20),
    (45 seconds, (assert, wait), st_1_7_20) ) )

(st_1_7_15, ( (7, clear),
   (7, write, "Very good.  You positioned the DETECTOR correctly.")
),
   st_7_100 )
```

```
(st_1_7_20, ( (7, clear),
   (4, write, "Do you need HELP in how to move the DETECTOR?"),
   (9, input, keyboard) ),
   ( (("YES" | "Yes" | "yes" | "y" | "Y"), (assert, help), st_1_5_1),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_10 miss_left) &
              ((past st_1_7_30 move_detector) | (past st_1_7_30 miss_again)),
               (assert, help), st_1_1_45),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_10 miss_left), st_1_7_25),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_10 miss_right) &
              ((past st_1_7_30 move_detector) | (past st_1_7_30 miss_again)),
               (assert, help), st_1_1_45),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_10 miss_right), st_1_7_35),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_10 miss_low) &
              ((past st_1_7_30 move_detector) | (past st_1_7_30 miss_again)),
               (assert, help), st_1_1_45),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_10 miss_low), st_1_7_40),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_10 miss_high) &
     ((past st_1_7_30 move_detector) | (past st_1_7_30 miss_again)),
     (assert, help), st_1_1_45),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_10 miss_high), st_1_7_45),
    (300 seconds, (assert, wait), st_1_1_60) ) )

(st_1_7_25, ( (7, clear),
   (4, write, "You moved the DETECTOR too far to the left."),
   (7, write, "Try again.") ),
   st_1_7_1 )

(st_1_7_30, ( (4, clear), (7, clear),
   (4, write, "To position the DETECTOR correctly, the RAY PATH must hit some part of
the DETECTOR."),
   (7, write, "Move the DETECTOR so that the SOUND will hit the DETECTOR."),
   (1, input, mouse) ),
   ( (click-left, (assert, move_detector), st_1_7_10),
    (assert, miss_again), st_1_7_20 ) )

(st_1_7_35, ( (7, clear),
   (4, write, "You moved the DETECTOR too far to the right."),
   (7, write, "Try again.") ),
   st_1_7_1 )
```

```
(st_1_7_40, ( (7, clear),
   (4, write, "You moved the DETECTOR too far to the low."),
   (7, write, "Try again.") ),
   st_1_7_1 )

(st_1_7_45, ( (7, clear),
   (4, write, "You moved the DETECTOR too far to the high."),
   (7, write, "Try again.") ),
   st_1_7_1 )

/* End TASK Direct Path */

/* TASK Reflected Surface Refracted (RSR) */

(st_7_100, ( (0, clear),
   (1, draw, source@(mouseX, mouseY) ),
   (1, draw, path@(mouseX, mouseY) ), /* RSR Path */
   (1, draw, detector@(mouseX, mouseY) ),
   (3, write, "Move the DETECTOR so that the SOUND will hit the DETECTOR."),
   (0, input, mouse),
   (9, input, keyboard) ),
   ( (click-left, (assert, move_detector), st_1_7_110),
    (45 seconds, (assert, wait), st_1_10_120),
    ("Help" | "I do not know" | "How", (assert, help), st_1_5_1),
    st_1_7_120) )

(st_1_7_110, ( (1, clear, detector@(detector.x, detector.y)),
   (1, draw, detector@(mouseX, mouseY)) ),
   ( (click-left & (mouseX <= sound_RSR.x + halfwid & mouseX >= sound_RSR.x -
halfwid & mouseY <= sound_RSR.y + halfht & mouseY >= sound_RSR.y - halfht),
(assert, hit), st_1_7_115),
    (click-left & ((past st_1_7_110 miss_left) | (past st_1_7_110 miss_right)
    | (past st_1_7_110 miss_low) | (past st_1_7_110 miss_high)),
                (assert, help), st_1_7_130),
    (click-left & (mouseX < sound_RSR.x - halfwid), (assert, miss_left), st_1_7_120),
    (click-left & (mouseX > sound_RSR.x + halfwid), (assert, miss_right), st_1_7_120),
    (click-left & (mouseY < sound_RSR.y - halfht), (assert, miss_low), st_1_7_120),
    (click-left & (mouseY > sound_RSR.y + halfht), (assert, miss_high), st_1_7_120),
    (45 seconds, (assert, wait), st_1_7_120) ) )
```

```
(st_1_7_115, ( (7, clear),
   (7, write, "Very good.  You positioned the DETECTOR correctly."),
   (0,quit)),
   st_1_7_115 )

(st_1_7_120, ( (7, clear),
   (4, write, "Do you need HELP in how to move the DETECTOR?"),
   (9, input, keyboard) ),
   ( (("YES" | "Yes" | "yes" | "y" | "Y"), (assert, help), st_1_5_1),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_110 miss_left) &
     ((past st_1_7_130 move_detector) | (past st_1_7_130 miss_again)),
                (assert, help), st_1_1_45),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_110 miss_left), st_1_7_125),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_110 miss_right) &
     ((past st_1_7_130 move_detector) | (past st_1_7_130 miss_again)),
                (assert, help), st_1_1_45),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_110 miss_right), st_1_7_135),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_110 miss_low) &
     ((past st_1_7_130 move_detector) | (past st_1_7_130 miss_again)),
                 (assert, help), st_1_1_45),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_110 miss_low), st_1_7_140),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_110 miss_high) &
                ((past st_1_7_130 move_detector) | (past st_1_7_130 miss_again)),
                 (assert, help), st_1_1_45),
    (("NO" | "No" | "no" | "n" | "N") & (past st_1_7_110 miss_high), st_1_7_145),
    (300 seconds, (assert, wait), st_1_1_60) ) )

(st_1_7_125, ( (7, clear),
   (4, write, "You moved the DETECTOR too far to the left."),
   (7, write, "Try again.") ),
   st_1_7_100 )

(st_1_7_130, ( (4, clear), (7, clear),
   (4, write, "To position the DETECTOR correctly, the RAY PATH must hit some part of
the DETECTOR."),
   (7, write, "Move the DETECTOR so that the SOUND will hit the DETECTOR."),
   (1, input, mouse) ),
   ( (click-left, (assert, move_detector), st_1_7_110),
    (assert, miss_again), st_1_7_120 ) )
```

```
(st_1_7_135, ( (7, clear),
   (4, write, "You moved the DETECTOR too far to the right."),
   (7, write, "Try again.") ),
   st_1_7_100 )

(st_1_7_140, ( (7, clear),
   (4, write, "You moved the DETECTOR too far to the low."),
   (7, write, "Try again.") ),
   st_1_7_100 )


(st_1_7_145, ( (7, clear),
   (4, write, "You moved the DETECTOR too far to the high."),
   (7, write, "Try again.") ),
   st_1_7_100 )

/* End TASK RSR */
```

APPENDIX C

GRAMMAR

```
cfd_graph ::= cfd_graph cfd_node | cfd_node | cfd_menu cfd_node
cfd_node ::= "(" cfd_id "," action_list "," response_list ")" | identifier ":=" string
cfd_menu ::= "(" "menu" string menulist ")"
menulist ::= string "->" cfd_id "," menulist | string "->" cfd_id
cfd_id ::= identifier
action_list ::= "(" act_node_list ")" | action_node
response_list ::= "(" res_node_list ")" | response_node
action_node ::= "(" region_id "," action ")"
act_node_list ::= act_node_list "," action_node | action_node
response_node ::= "(" pattern "," cfd_id ")
   | "(" pattern "," "(" "assert" "," identifier ")" "," cfd_id ")"
   | "(" pattern "," "ignore" ")"
   | "(" pattern "," "(" "assert" "," identifier ")" "," "ignore" ")"
   | "(" "assert" "," identifier ")" "," cfd_id | cfd_id
res_node_list ::= res_node_list "," response_node
   | response_node
region_id ::= integer | region_id "+" integer
action ::= "draw" "," identfier | "draw" "," identifier "@" location | "clear"
   | "clear" "," identifier "@" location | "write" "," string | "input" "," input_list
   | "pause" "," integer | "drag" "," identifier | "quit"
input_list ::= "mouse" | "keyboard" | "mouse&key"
location ::= "(" loc_part "," loc_part ")"
loc_part ::= loc_part "+" term | loc_part "-" term | term
term ::= term "*" factor | term "/" factor | factor
factor ::= integer | "mouseX" | "mouseY" | identifier ".x" | identifier ".y" | "halfwid"
   | "halfht" | "(" loc_part ")"
patpart ::= keywords | "click-left" | "click-right" | "click-middle" | "click-any"
   | loc_part relop loc_part | "click-exit" | "click-help" | "click-continue"
   | "mouse-move" | integer "seconds" | "past" cfd_id identifier | "(" pattern ")"
patconj ::= patpart | patconj "&" patpart
pattern ::= patconj | pattern "|" patconj
relop ::= "==" | ">" | "<" | ">=" | "<="
keywords ::= string
```

APPENDIX D

lex AND yacc FILES

A. parser.h - HEADER FILE

```
/* defines for cfd data structures and values */

/* region codes, or'd to make resultant code */
#define REG_ALL    0
#define REG_ONE    1
#define REG_TWO    2
#define REG_THREE  4
#define REG_FOUR   8
#define REG_FIVE   16
#define REG_SIX    32
#define REG_SEVEN  64
#define REG_EIGHT 128
#define REG_NINE  256
#define REG_OTHER 512

/* action codes */
#define ACT_NULL  0
#define ACT_DRAW  1
#define ACT_CLEAR 2
#define ACT_WRITE 3
#define ACT_INPUT 4
#define ACT_PAUSE 5
#define ACT_QUIT  6
#define ACT_DRAG  7

/* input modes */
#define MODE_NULL     0
#define MODE_MOUSEONLY 1
#define MODE_KEYONLY  2
#define MODE_MOUSEKEY 3
```

```
/* response codes */
#define RES_NULL      0
#define RES_KEY       1
#define RES_CLICKLEFT 2
#define RES_CLICKRIGHT 3
#define RES_CLICKMID  4
#define RES_CLICKANY  5
#define RES_MOUSEX    6 /* unused */
#define RES_MOUSEY    7 /* unused */
#define RES_MOUSEMOVE 8
#define RES_SECONDS   9
#define RES_PAST      10
#define RES_CLICKHELP 11
#define RES_CLICKCONT 12
#define RES_CLICKEXIT 13

/* operator codes */
#define OP_NULL       0
#define OP_GREAT      1
#define OP_GEQ        2
#define OP_EQ         3
#define OP_LEQ        4
#define OP_LESS       5
#define OP_AND        6
#define OP_OR         7
#define OP_KEYOR      8 /* unused */

struct expnode {
 struct expnode *left, *right;
 char op;      /* only checked if left or right isn't null */
 char *varname; /* mousex, mousey, or graph id */
 char comp;    /* x, y or blank */
 int val;      /* integer value in expression */
};

struct locnode {
 struct expnode *x, *y;
} ;
```

```
struct actnode {
 int actloc;    /* region codes or'd together */
 int action;    /* action code */
 char *info_str; /* string arguments to action, including graph filename */
 int  info_int;  /* integer arguments to action, including input mode */
 struct locnode *info_loc;  /* location arguments to action */
 struct actnode *next; /* next action list */
 };

struct opnode {
 struct opnode *left; /* left arg to operator */
 int operator;   /* operator code */
 struct opnode *right; /* right arg to operator */
 int res_act;    /* response code */
 struct expnode *res_left, *res_right; /* expression arguments to response */
 int res_int;    /* integer argument to response, including seconds */
 char *res_str;  /* string argument to response , includeing keywords */
 };

struct resnode {
 struct opnode *expr;  /* expression tree to match response */
 char *label;          /* assertion label */
 struct cfdnode *node; /* node to go to if match response, null to ignore */
 struct resnode *next; /* next response option */
 };

struct cfdnode {
 char *nodeid;
 struct actnode  *actlist;
 struct resnode  *reslist;
 struct cfdnode *next;
};

struct menunode {
 char *choice;
 struct cfdnode *state;
 struct menunode *next;
 };
```

```
struct menu {
  char *title;
  struct menunode *choices;
  };

#ifdef MAIN
#define EXTERN
#else
#define EXTERN extern
#endif

EXTERN struct cfdnode *cfdgraph;
EXTERN struct cfdnode *cfdlist;
EXTERN struct menu *topmenu;

struct picnode {
 char *picid;
 char *picfile;
 struct picnode *next;
};
EXTERN struct picnode *piclist;

struct cfdnode *findnode();
```

B. parser.l - LEX

```
%{
#include <stdio.h>
#include "y.tab.h"
int yylineno;
int yylen;
int intval;
static int is_comm = 0;
char *malloc();
%}

%%
[ \t]+                  ;
#[^\n]*                 ;
\n                      {yylineno++; };
\/\*                    {is_comm = 1; }
\*\/                    {is_comm = 0; }
\+                      {if (!is_comm) {return(TOK_ADD);}} ;
-                       {if (!is_comm) {return(TOK_SUBTRACT);}} ;
\*                      {if (!is_comm) {return(TOK_TIMES);}} ;
\/                      {if (!is_comm) {return(TOK_DIVIDE);}} ;
@                       {if (!is_comm) {return(TOK_AT);}} ;
\|                      {if (!is_comm) {return(TOK_BAR);}} ;
\,                      {if (!is_comm) {return(TOK_COMMA);}} ;
\(                      {if (!is_comm) {return(TOK_OPENPAREN);}} ;
\)                      {if (!is_comm) {return(TOK_CLOSEPAREN);}};
\>=                     {if (!is_comm) {return(TOK_GREATEQ);}} ;
\<=                     {if (!is_comm) {return(TOK_LESSEQ);}};
==                      {if (!is_comm) {return(TOK_EQUAL);}};
\<                      {if (!is_comm) {return(TOK_LESS);}} ;
-\>                     {if (!is_comm) {return(TOK_ARROW);}} ;
\:=                     {if (!is_comm) {return(TOK_DEFINE);}} ;
\>                      {if (!is_comm) {return(TOK_GREATER);}} ;
\"[^\"]*\"      {if (!is_comm) {
            yylen = strlen(yytext);
            yylval.t_str = malloc(yylen+1);
            strncpy(yylval.t_str,&(yytext[1]),yylen-2);
            return(TOK_STR);}} ;
```

```
[0-9]+                          {if (!is_comm) {
            intval = 0;
            yylen = strlen(yytext);
            yylval.t_str = malloc(yylen+1);
            strncpy(yylval.t_str,yytext,yylen);
            while (is_comm < yylen) {
                                    intval = intval*10 + yytext[is_comm] - '0';
                                    is_comm++;
                                    }
                                    is_comm = 0;
                                   return(TOK_NUM);
                                   } } ;
ignore                             {if (!is_comm) {return(TOK_IGNORE);}} ;
drag                               {if (!is_comm) {return(TOK_DRAG);}} ;
draw                               {if (!is_comm) {return(TOK_DRAW);}} ;
clear                              {if (!is_comm) {return(TOK_CLEAR);}} ;
write                              {if (!is_comm) {return(TOK_WRITE);}} ;
input                              {if (!is_comm) {return(TOK_INPUT);}} ;
pause                              {if (!is_comm) {return(TOK_PAUSE);}} ;
assert                             {if (!is_comm) {return(TOK_ASSERT);}} ;
mouse                              {if (!is_comm) {return(TOK_MOUSE);}} ;
keyboard        {if (!is_comm) {return(TOK_KEY);}} ;
mouse\&key      {if (!is_comm) {return(TOK_MOUSEKEY);}} ;
click-left     {if (!is_comm) {return(TOK_CLICKLEFT);}} ;
click-right     {if (!is_comm) {return(TOK_CLICKRIGHT);}};
click-middle    {if (!is_comm) {return(TOK_CLICKMID);}} ;
click-any       {if (!is_comm) {return(TOK_CLICKANY);}} ;
click-help      {if (!is_comm) {return(TOK_CLICKHELP);}};
click-continue {if (!is_comm) {return(TOK_CLICKCONT);}} ;
click-exit      {if (!is_comm) {return(TOK_CLICKEXIT);}} ;
halfwid                            {if (!is_comm) {return(TOK_HALFWID);}} ;
halfht                             {if (!is_comm) {return(TOK_HALFHT);}} ;
mouseX                             {if (!is_comm) {return(TOK_MOUSEX);}} ;
mouseY                             {if (!is_comm) {return(TOK_MOUSEY);}} ;
mouse-move      {if (!is_comm) {return(TOK_MOUSEMOVE);}} ;
seconds                            {if (!is_comm) {return(TOK_SECONDS);}} ;
past                               {if (!is_comm) {return(TOK_PAST);}} ;
menu                               {if (!is_comm) {return(TOK_MENU);}} ;
quit                               {if (!is_comm) {return(TOK_QUIT);}} ;
\x                                 {if (!is_comm) {return(TOK_XCOMP);}} ;
\y          {if (!is_comm) {return(TOK_YCOMP);}} ;
```

```
[0-9A-Za-z_-]+{if (!is_comm) {
            yylen = strlen(yytext);
            yylval.t_str = malloc(yylen+1);
            strncpy(yylval.t_str,yytext,yylen);
            return(TOK_ID);}} ;
\&          {if (!is_comm) {return(TOK_AMPERSAND);}} ;
.                           {if (!is_comm)
            fprintf(stderr,"unrecognized '%s'\n",yytext);} ;
```

C. parser.y - YACC

```
%union {
 char *t_str;
 int  t_int;
 struct cfdnode *t_cfd;
 struct actnode *t_act;
 struct resnode *t_res;
 struct opnode *t_op;
 struct locnode *t_loc;
 struct expnode *t_exp;
 struct menunode *t_men;
 }

%token <t_int> TOK_OPENPAREN TOK_CLOSEPAREN
%token <t_int> TOK_EQUAL TOK_GREATER TOK_LESS TOK_GREATEQ
TOK_LESSEQ
%token <t_int> TOK_ADD TOK_SUBTRACT TOK_TIMES TOK_DIVIDE
%token <t_int> TOK_BAR TOK_AMPERSAND
%token <t_int> TOK_COMMA TOK_DEFINE TOK_MENU TOK_ARROW
%token <t_int> TOK_NUM
%token <t_int> TOK_ID
%token <t_int> TOK_STR
%token <t_int> TOK_IGNORE
%token <t_int> TOK_DRAW TOK_WRITE TOK_CLEAR TOK_INPUT TOK_PAUSE
TOK_ASSERT
%token <t_int> TOK_DRAG TOK_QUIT TOK_MOUSEKEY TOK_MOUSE
TOK_KEY
%token <t_int> TOK_CLICKLEFT TOK_CLICKRIGHT TOK_CLICKMID
TOK_CLICKANY
%token <t_int> TOK_CLICKHELP TOK_CLICKCONT TOK_CLICKEXIT
%token <t_int> TOK_MOUSEX TOK_MOUSEY TOK_MOUSEMOVE TOK_PAST
%token <t_int> TOK_SECONDS TOK_HALFWID TOK_HALFHT
%token <t_int> TOK_XCOMP TOK_YCOMP TOK_AT
%type <t_str> cfd_id
%type <t_men> menu_list
%type <t_men> choice
%type <t_act> action_list
%type <t_act> action_node
%type <t_act> act_node_list
%type <t_res> response_list
%type <t_res> response_node
%type <t_str> exp_assert
%type <t_res> res_node_list
```

```
%type <t_int> region_id
%type <t_act> action
%type <t_int> input_list
%type <t_op> keywords
%type <t_op> pattern
%type <t_op> patpart
%type <t_op> patconj
%type <t_int> relop
%type <t_loc> location
%type <t_exp> locpart
%type <t_exp> term
%type <t_exp> factor

%start cfd_graph

%%
cfd_graph : cfd_graph cfd_node | cfd_graph cfd_def |
        start cfd_node | start cfd_def |
        start cfd_menu cfd_node | start cfd_menu cfd_def ;

start: {
 cfdlist = NULL;
 cfdgraph = NULL;
 piclist = NULL;
 topmenu = NULL;
#ifdef YYDEBUG
#if YYDEBUG
yydebug = 1;
#else
yydebug = 0;
#endif
#endif
} ;

cfd_menu: TOK_OPENPAREN TOK_MENU TOK_STR
      {topmenu = NEWPTR(menu);
       topmenu->title = yylval.t_str;
      } menu_list TOK_CLOSEPAREN
      {topmenu->choices = $5;} ;
```

```
menu_list: choice TOK_COMMA menu_list
        {tmp_menu = $1;
                    tmp_menu->next = $3;
                    $$ = tmp_menu;
                   } | choice {$$ = $1;};

choice: TOK_STR {tmp_str = yylval.t_str;} TOK_ARROW cfd_id
                {tmp_menu = NEWPTR(menunode);
       tmp_menu->choice = tmp_str;
       tmp_menu->state = findnode($4);
                    tmp_menu->next = NULL;
       $$ = tmp_menu;
                    };

cfd_node : TOK_OPENPARENcfd_id TOK_COMMA action_list  TOK_COMMA
        response_list TOK_CLOSEPAREN
                                {tmp_node = findnode($2);
            tmp_node->actlist = $4;
            tmp_node->reslist = $6;
            if (cfdgraph == NULL) cfdgraph = tmp_node;
            /* else it's already linked into the graph */
           } ;

cfd_def : TOK_ID {tmp_str = yylval.t_str;} TOK_DEFINE TOK_STR
                {add_defn(tmp_str,yylval.t_str);} :

cfd_id : TOK_ID { $$ = my_copy(yylval.t_str,yylen); } ;

action_list : TOK_OPENPAREN act_node_list TOK_CLOSEPAREN {$$ = $2;}
         | action_node {$$ = $1;} :

response_list : TOK_OPENPAREN res_node_list TOK_CLOSEPAREN {$$ = $2;}
         | response_node {$$ = $1;} ;

action_node : TOK_OPENPAREN region_id TOK_COMMA action
TOK_CLOSEPAREN
         { tmp_act = $4;
                                tmp_act->actloc = $2;
                                tmp_act->next = NULL;
                                $$ = tmp_act:
                    };
```

```
act_node_list : act_node_list TOK_COMMA action_node
                {tmp_act = $1;
                 if (tmp_act == NULL) $$ = $3;
                 else {
                        while (tmp_act->next != NULL) tmp_act=tmp_act->next;
                        tmp_act->next = $3;
                        $$ = $1;
                        }
        }
       | action_node {$$ = $1;} ;

response_node : TOK_OPENPAREN pattern TOK_COMMA cfd_id
TOK_CLOSEPAREN
                {tmp_res = NEWPTR(resnode);
        tmp_res->label = NULL;
                 tmp_res->expr = $2;
                 tmp_res->node = findnode($4);
                 $$ = tmp_res;
                }
                | TOK_OPENPAREN pattern TOK_COMMA exp_assert
TOK_COMMA cfd_id
         TOK_CLOSEPAREN
                {tmp_res = NEWPTR(resnode);
        tmp_res->label = $4;
                 tmp_res->expr = $2;
                 tmp_res->node = findnode($6);
                 $$ = tmp_res;
        }
                | TOK_OPENPAREN pattern TOK_COMMA exp_assert
TOK_COMMA TOK_IGNORE
         TOK_CLOSEPAREN
                {tmp_res = NEWPTR(resnode);
        tmp_res->label = $4;
                 tmp_res->expr = $2;
                 tmp_res->node = NULL;
                 $$ = tmp_res;
                }
                | TOK_OPENPAREN pattern TOK_COMMA TOK_IGNORE
TOK_CLOSEPAREN
                {tmp_res = NEWPTR(resnode);
        tmp_res->label = NULL;
                 tmp_res->expr = $2;
                 tmp_res->node = NULL;
```

```
                $$ = tmp_res;
                }
                | exp_assert TOK_COMMA cfd_id
                { tmp_res = NEWPTR(resnode);
                 tmp_res->expr = NULL;
        tmp_res->label = $1;
                 tmp_res->next = NULL;
                 tmp_res->node = findnode($3);
                 $$ = tmp_res;
                }
                | cfd_id
                { tmp_res = NEWPTR(resnode);
                 tmp_res->expr = NULL;
        tmp_res->label = NULL;
                 tmp_res->next = NULL;
                 tmp_res->node = findnode($1);
                 $$ = tmp_res;
                } ;

exp_assert: TOK_OPENPARAN TOK_ASSERT TOK_COMMA TOK_ID
     {tmp_str = my_copy(yylval.t_str,yylen);} TOK_CLOSEPARAN
     {$$ = tmp_str;} ;

res_node_list : res_node_list TOK_COMMA response_node
                {tmp_res = $1;
        $$ = NULL;
                 if (tmp_res == NULL) $$ = $3;
                 else {
                                while(tmp_res->next != NULL) tmp_res = tmp_res->next;
         tmp_res->next = $3;
                                $$ = $1;
                 }
                }
                | response_node {$$ = $1;} ;
```

```
region_id : TOK_NUM
                {switch (intval) {
                 case 0 : $$ = REG_ALL; break;
                 case 1 : $$ = REG_ONE; break;
                 case 2 : $$ = REG_TWO; break;
                 case 3 : $$ = REG_THREE; break;
                 case 4 : $$ = REG_FOUR; break;
                 case 5 : $$ = REG_FIVE; break;
                 case 6 : $$ = REG_SIX; break;
                 case 7 : $$ = REG_SEVEN; break;
                 case 8 : $$ = REG_EIGHT; break;
                 case 9 : $$ = REG_NINE; break;
                 default: $$ = REG_OTHER;
                 }
                }
                | region_id TOK_ADD TOK_NUM
                {switch (intval) {
          case 0 : $$ = $1 | REG_ALL; break;
          case 1 : $$ = $1 | REG_ONE; break;
          case 2 : $$ = $1 | REG_TWO; break;
          case 3 : $$ = $1 | REG_THREE; break;
          case 4 : $$ = $1 | REG_FOUR; break;
          case 5 : $$ = $1 | REG_FIVE; break;
          case 6 : $$ = $1 | REG_SIX; break;
          case 7 : $$ = $1 | REG_SEVEN; break;
          case 8 : $$ = $1 | REG_EIGHT; break;
          case 9 : $$ = $1 | REG_NINE; break;
          default: $$ = $1 | REG_OTHER;
          }
         } ;
```

```
action : TOK_DRAW TOK_COMMA TOK_ID
                {if ((t_pic=findpic(yylval.t_str))!=NULL)
                 $$ = new_actnode(ACT_DRAW, t_pic->picfile, 0, NULL);
                 else {
                  fprintf(stderr,"Warning: undefined graph %s\n",yylval.t_str);
                  $$ = new_actnode(ACT_DRAW, yylval.t_str, 0, NULL);
                  } }
     | TOK_DRAW TOK_COMMA TOK_ID {tmp_str = yylval.t_str;} TOK_AT location
                {if ((t_pic = findpic(tmp_str))!=NULL)
                 $$ = new_actnode(ACT_DRAW, t_pic->picfile, 0, $6);
                 else {
                  fprintf(stderr,"Warning: undefined graph %s\n",tmp_str);
                  $$ = new_actnode(ACT_DRAW, tmp_str, 0, $6);
                  } }
      | TOK_DRAG TOK_COMMA TOK_ID
       {if ((t_pic = findpic(yylval.t_str))!=NULL)
         $$ = new_actnode(ACT_DRAG, t_pic->picfile, 0, NULL);
        else {
         fprintf(stderr,"Warning: undefined graph %s\n",yylval.t_str);
         $$ = new_actnode(ACT_DRAG, yylval.t_str, 0, NULL);
         } }
                | TOK_CLEAR
                  {$$ = new_actnode(ACT_CLEAR,NULL, 0, NULL);}
                | TOK_QUIT
                  {$$ = new_actnode(ACT_QUIT,NULL, 0, NULL);}
                | TOK_CLEAR TOK_COMMA TOK_ID {tmp_str = yylval.t_str;}
TOK_AT location
                {if ((t_pic = findpic(tmp_str))!=NULL)
                 $$ = new_actnode(ACT_CLEAR, t_pic->picfile, 0, $6);
                 else {
                  fprintf(stderr,"Warning: undefined graph %s\n",tmp_str);
                  $$ = new_actnode(ACT_CLEAR, tmp_str, 0, $6);
                  } }
                | TOK_WRITE TOK_COMMA TOK_STR
                  {$$ = new_actnode(ACT_WRITE,yylval.t_str,0,NULL);}
                | TOK_INPUT TOK_COMMA input_list
                  {$$ = new_actnode(ACT_INPUT,NULL,$2,NULL);}
                | TOK_PAUSE TOK_COMMA TOK_NUM
                  {$$ = new_actnode(ACT_PAUSE,NULL, intval, NULL);}
                ;
```

```
location : TOK_OPENPAREN locpart TOK_COMMA locpart TOK_CLOSEPAREN
                {tmp_loc = NEWPTR(locnode);
                 tmp_loc->x = $2;
                 tmp_loc->y = $4;
                 $$ = tmp_loc;
                 } ;

locpart : term {$$ = $1;}
                | locpart TOK_ADD term
                {tmp_exp = NEWPTR(expnode);
                 tmp_exp->left = $1;
                 tmp_exp->op = '+';
                 tmp_exp->right = $3;
                 tmp_exp->varname = NULL;
                 tmp_exp->comp = ' ';
                 tmp_exp->val = 0;
        $$ = tmp_exp; }
                | locpart TOK_SUBTRACT term
                {tmp_exp = NEWPTR(expnode);
                 tmp_exp->left = $1;
                 tmp_exp->op = '-';
                 tmp_exp->right = $3;
                 tmp_exp->varname = NULL;
                 tmp_exp->comp = ' ';
                 tmp_exp->val = 0;
        $$ = tmp_exp; } ;

term : factor { $$ = $1; }
     | term TOK_TIMES factor
                {tmp_exp = NEWPTR(expnode);
                 tmp_exp->left = $1;
                 tmp_exp->op = '*';
                 tmp_exp->right = $3;
                 tmp_exp->varname = NULL;
                 tmp_exp->comp = ' ';
                 tmp_exp->val = 0;
                 $$ = tmp_exp; }
     | term TOK_DIVIDE factor
                {tmp_exp = NEWPTR(expnode);
                 tmp_exp->left = $1;
                 tmp_exp->op = '/';
                 tmp_exp->right = $3;
                 tmp_exp->varname = NULL;
```

```
                tmp_exp->comp = ' ';
                tmp_exp->val = 0;
                $$ = tmp_exp; } ;

factor : TOK_NUM
                {tmp_exp = NEWPTR(expnode);
                tmp_exp->left = NULL;
                tmp_exp->op = ' ';
                tmp_exp->right = NULL;
                tmp_exp->varname = NULL;
                tmp_exp->comp = ' ';
                tmp_exp->val = intval;
                $$ = tmp_exp; }
        | TOK_HALFWID
                {tmp_exp = NEWPTR(expnode);
                tmp_exp->left = NULL;
                tmp_exp->op = ' ';
                tmp_exp->right = NULL;
                tmp_exp->varname = "halfwid";
                tmp_exp->comp = ' ';
                tmp_exp->val = 0;
                $$ = tmp_exp; }
        | TOK_HALFHT
                {tmp_exp = NEWPTR(expnode);
                tmp_exp->left = NULL;
                tmp_exp->op = ' ';
                tmp_exp->right = NULL;
                tmp_exp->varname = "halfht";
                tmp_exp->comp = ' ';
                tmp_exp->val = 0;
                $$ = tmp_exp; }
        | TOK_MOUSEX
                {tmp_exp = NEWPTR(expnode);
                tmp_exp->left = NULL;
                tmp_exp->op = ' ';
                tmp_exp->right = NULL;
                tmp_exp->varname = "mouseX";
                tmp_exp->comp = ' ';
                tmp_exp->val = 0;
                $$ = tmp_exp; }
        | TOK_MOUSEY
                {tmp_exp = NEWPTR(expnode);
                tmp_exp->left = NULL;
```

```
                tmp_exp->op = ' ';
                tmp_exp->right = NULL;
                tmp_exp->varname = "mouseY";
                tmp_exp->comp = ' ';
                tmp_exp->val = 0;
                $$ = tmp_exp; }
              | TOK_ID TOK_XCOMP
                {tmp_exp = NEWPTR(expnode);
                tmp_exp->left = NULL;
                tmp_exp->op = ' ';
                tmp_exp->right = NULL;
                tmp_exp->varname = yylval.t_str;
                tmp_exp->comp = 'x';
                tmp_exp->val = 0;
                $$ = tmp_exp; }
              | TOK_ID TOK_YCOMP
                {tmp_exp = NEWPTR(expnode);
                tmp_exp->left = NULL;
                tmp_exp->op = ' ';
                tmp_exp->right = NULL;
                tmp_exp->varname = yylval.t_str;
                tmp_exp->comp = 'y';
                tmp_exp->val = 0;
                $$ = tmp_exp; }
              | TOK_OPENPAREN locpart TOK_CLOSEPAREN {$$ = $2;};

input_list : TOK_MOUSE {$$ = MODE_MOUSEONLY;}
           | TOK_KEY {$$ = MODE_KEYONLY;}
           | TOK_MOUSEKEY {$$ = MODE_MOUSEKEY;} ;

patpart : keywords {$$ = $1;}
           | TOK_CLICKLEFT
      {$$ = new_opnode(OP_NULL,NULL,NULL,NULL,NULL,
                                    RES_CLICKLEFT,0,NULL);}
           | TOK_CLICKRIGHT
      {$$ = new_opnode(OP_NULL,NULL,NULL,NULL,NULL,
                                    RES_CLICKRIGHT,0,NULL);}
           | TOK_CLICKMID
      {$$ = new_opnode(OP_NULL,NULL,NULL,NULL,NULL,
                                    RES_CLICKMID,0,NULL);}
           | TOK_CLICKANY
      {$$ = new_opnode(OP_NULL,NULL,NULL,NULL,NULL,
                                    RES_CLICKANY,0,NULL);}
```

```
                | TOK_CLICKHELP
        {$$ = new_opnode(OP_NULL,NULL,NULL,NULL,NULL,
                                        RES_CLICKCONT,0,NULL);}
                | TOK_CLICKCONT
        {$$ = new_opnode(OP_NULL,NULL,NULL,NULL,NULL,
                                        RES_CLICKCONT,0,NULL);}
                | TOK_CLICKEXIT
                  {$$ = new_opnode(OP_NULL,NULL,NULL,NULL,NULL,
                                        RES_CLICKEXIT,0,NULL);}
                | locpart relop locpart
                  {$$ = new_opnode($2,NULL,NULL,$1,$3,
                                        RES_NULL,0,NULL);}
                | TOK_MOUSEMOVE
                  {$$ = new_opnode(OP_NULL,NULL,NULL,NULL,NULL,
                                        RES_MOUSEMOVE,0,NULL);}
                | TOK_NUM TOK_SECONDS
                  {$$ = new_opnode(OP_NULL,NULL,NULL,NULL,NULL,
                                        RES_SECONDS,intval,NULL);}
                | TOK_PAST cfd_id TOK_ID
                  {
                   tmp_str = malloc(strlen($2)+yylen+2);
                   strcpy(tmp_str,$2);
                   strcat(tmp_str,"/");
        strcat(tmp_str,yylval.t_str);
                   $$ =
new_opnode(OP_NULL,NULL,NULL,NULL,NULL,RES_PAST,0,tmp_str); }
                | TOK_OPENPAREN pattern TOK_CLOSEPAREN
                 {$$ = $2;}
                 ;

patconj : patpart {$$ = $1;}
                | patconj TOK_AMPERSAND patpart
                 { $$ =
new_opnode(OP_AND,$1,$3,NULL,NULL,RES_NULL,0,NULL); } ;

pattern : patconj {$$ = $1;} | pattern TOK_BAR patconj
                 { $$ = new_opnode(OP_OR,$1,$3,NULL,NULL,RES_NULL,0,NULL);
} ;
```

```
relop : TOK_EQUAL {$$ = OP_EQ;}
      | TOK_GREATER {$$ = OP_GREAT;}
                | TOK_LESS {$$ = OP_LESS;}
                | TOK_GREATEQ {$$ = OP_GEQ;}
                | TOK_LESSEQ {$$ = OP_LEQ;} ;

keywords : TOK_STR
                  {$$ =
new_opnode(OP_NULL,NULL,NULL,NULL,NULL,RES_KEY,0,yylval.t_str);}
                  ;

%%
#include <stdio.h>
#ifdef STANDALONE
#define MAIN
#include "parser.h"
#undef MAIN
#else
#include "parser.h"
#endif

extern int yylineno;
extern char *yytext;
extern int yylen;
extern int intval;
char *malloc();
char *tmp_str;
struct expnode *tmp_exp;
struct opnode *tmp_op;
struct resnode *tmp_res;
struct cfdnode *tmp_node;
struct actnode *tmp_act;
struct locnode *tmp_loc;
struct menunode *tmp_menu;
struct picnode *t_pic;

#define NEWPTR(Type) (struct Type *) malloc(sizeof (struct Type))
```

```
char *my_copy(str,len)
char *str;
int len;
{char *tmp;
 if (str == NULL || len <= 0) {
  fprintf(stderr,"null string sent to my_copy\n");
  return NULL;
  }
 else {
  if ((tmp = malloc(len+1)) == NULL) {
   fprintf(stderr,"Cannot allocate memory for string of length %d\n",len+1);
   exit(2);
   }
  strncpy(tmp, str,len);
#if YYDEBUG
  fprintf(stdout,"copied %s (%d)\n",tmp,len);
#endif
  return tmp;
  }
}

struct opnode *new_opnode(op,oleft,oright,rleft,rright,act,num,str)
struct opnode *oleft, *oright;
int op,num,act;
struct expnode *rleft, *rright;
char *str;
{struct opnode *tmp = NEWPTR(opnode);
 tmp->left = oleft;
 tmp->operator = op;
 tmp->right = oright;
 tmp->res_act = act;
 tmp->res_left = rleft;
 tmp->res_right = rright;
 tmp->res_int = num;
 tmp->res_str = str;
 return tmp;
}
```

```
struct actnode *new_actnode(actcode, str, num, loc)
int actcode, num;
char *str;
struct locnode *loc;
{struct actnode *tmp = NEWPTR(actnode);
 tmp->actloc = 0; /* whole screen, by default */
 tmp->action = actcode;
 tmp->info_str = str;
 tmp->info_int = num;
 tmp->info_loc = loc;
 tmp->next = NULL;
 return tmp;
}

struct cfdnode *findnode(target)
char *target;
{struct cfdnode *cur = cfdlist;
 if (cur == NULL) {
  cfdlist = NEWPTR(cfdnode);
  cfdlist->nodeid = target;
  cfdlist->actlist = NULL;
  cfdlist->reslist = NULL;
  cfdlist->next = NULL;
  return cfdlist;
  }
 else {
  if (target == NULL) return cfdlist;
  while (cur->next != NULL && strcmp(cur->nodeid,target)!=0) cur = cur->next;
  if (strcmp(cur->nodeid,target)==0) return cur;
  else {
   cur->next = NEWPTR(cfdnode);
   cur = cur->next;
   cur->nodeid = target;
   cur->actlist = NULL;
   cur->reslist = NULL;
   cur->next = NULL;
   return cur;
   }
  }
  return NULL; /* should be unreachable, but let's be safe! */
}
```

```
struct picnode *findpic(target)
char *target;
{struct picnode *cur = piclist;
 if (cur == NULL || target == NULL) return NULL;
 while (cur->next != NULL && strcmp(cur->picid,target) != 0) cur = cur->next;
 if (strcmp(cur->picid,target) == 0) return cur;
 else return NULL;
}

void add_defn(id,filename)
char *id, *filename;
{struct picnode *cur = piclist;
 if (id == NULL || filename == NULL) {
  fprintf(stderr,"Warning: null ident/filename in def on line %d\n",yylineno);
  return;
  }
 if (cur == NULL) {
  piclist = NEWPTR(picnode);
  piclist->picid = id;
  piclist->picfile = filename;
  piclist->next = NULL;
  }
 else {
  while (cur->next != NULL && strcmp(cur->picid,id) != 0) cur = cur->next;
  if (strcmp(cur->picid,id) == 0) {
   fprintf(stderr,"Warning: ignoring duplicate picture named %s on line %d\n",
                  id,yylineno);
   }
  else {
   cur->next = NEWPTR(picnode):
   cur = cur->next;
   cur->picid = id;
   cur->picfile = filename;
   cur->next = NULL;
  }
  }
}
```

```
int yyerror(s)
char *s;
{
 fflush(stdout);
 fflush(stderr);
 fprintf(stderr,"%s on line %d\n",s, yylineno);
}

#ifdef STANDALONE
main()
{
 if (yyparse()==0) printf("Successful parse\n");
 else printf("Unsucessful parse\n");
}
#endif
```

APPENDIX E

INTERPRETER

```
/*****************************************************************/
/* FILENAME: interp.h */
/* PURPOSE: Declaration of all global variables, function */
/* declarations and include files */
/* CALLED BY: interp.c */
/* AUTHOR: Dawn M. Maskell */
/* Timothy J. Shimeall */
/* Naval Postgraduate School, Monterey, CA */
/* DATE: 20 January 1991 */
/*****************************************************************/

/* include files */

#include <stdio.h>
#include <string.h>

#include "parser.h"

#include <suntool/sunview.h>
#include <suntool/canvas.h>
#include <suntool/icon.h>
#include <suntool/panel.h>
#include <suntool/alert.h>
#include <sunwindow/notify.h>
#include <sys/time.h>
```

```
#ifdef MAIN
#define EXTERN
#define INIT(Value) = Value
#else
#define EXTERN extern
#define INIT(Value)
#endif

#define ITIMER_NULL ((struct itimerval *)0)

/* set up conversion macros */

#define LINE_TO_RASTER_Y(Line) Line*16
#define RASTER_Y_TO_LINE(Ry) Ry/16
#define CHAR_TO_RASTER_X(Char) Char*8
#define RASTER_X_TO_CHAR(Rx) Rx/8

/* STRUCTURE declarations */

struct assert{ /* for assert list */
 char *id; /* state identifier being asserted */
 struct assert *next; /* ptr to next in assert list */
};
```

```
/* GLOBAL variables */

EXTERN struct assert *assert_list INIT(NULL); /* create assert list */

EXTERN struct cfdnode *current_state INIT(NULL);/* pointer to current*/
 /* state in tutorial */
 /* being executed */

EXTERN Canvas canvas; /* base canvas for display of */
 /* graphics and text */

EXTERN FILE *yyin; /* yacc input file, default stdin */

EXTERN Frame frame; /* base window of system */

EXTERN Icon post_icon;

EXTERN int panel_value; /* value of start state selection */
 timer_on; /* need to use a time */

EXTERN Panel menu_panel, /* display main menu buttons */
 panel; /* display base window panel buttons */

EXTERN Panel_item button, /* base window panel */
 cont, /* base window panel item button */
 help, /* base window panel item button */
 menu, /* base window main menu check boxes */
 quit; /* base window panel item button */

#ifdef MAIN
/* ICON declarations */
static short icon_image[] = {
#include "post_sym.icon"
};

EXTERN mpr_static(post_sym, 64, 64, 1, icon_image);
#endif
```

```
/* FUNCTION declarations */

EXTERN struct assert
 *create_assert_list();

EXTERN char
 *menu_selection();

EXTERN void
 canvas_repaint(),
 create_base_canvas(),
 create_base_frame(),
 create_base_panel(),
 do_action_list(),
 exit_selected(),
 kill_window(),
 get_file(),
 panel_repaint(),
 pause(),
 input_events(),
 present_concept();

EXTERN int
 continue_selected(),
 find_assert_list_id(),
 help_selected(),
 do_response_list(),
 start_selected();

EXTERN Notify_value
 pause_time(),
 my_notify();
```

```
/*************************************************************************/
/* FILENAME: interp.c */
/* PURPOSE: contains the routines to interpret concept-flow */
/* structure of the Physics of Sound Tutorial */
/* CONTAINS: main() */
/* get_file() */
/* create_base_frame() */
/* create_base_panel() */
/* create_base_canvas() */
/* AUTHORS: Dawn M. Maskell, LT, USN */
/* Timothy J. Shimeall */
/* Naval Postgraduate School, Monterey, CA */
/* DATE: 21 January 1992 */
/* SOFTWARE: SunView */
/*************************************************************************/

#define MAIN

#include "interp.h"

main(argc, argv)
int argc;
char *argv[];
{
 /* get the correct input file for tuturial */
 get_file(&argc, argv);

 /* Successful parse of file, execute tutorial */
 if (yyparse() == 0)
 {
 /* create base frame */
 create_base_frame(argc, argv);

 /* create base panel */
 create_base_panel();

 /* create base canvas */
 create_base_canvas();

 window_main_loop(frame);
 }
 /* Unsuccessful parse */
 else printf("Unsuccessful parse!\n");
```

```
/*********************************************************************/
/* FUNCTION: get_file() */
/* PURPOSE: get the name of the file to be used for tutorial */
/* CALLED BY: main() */
/* RETURNS: void */
/* CALLS: */
/* GLOBALS: FILE yyin <interp.h> */
/*********************************************************************/

void get_file(argc, argv)
int *argc;
char *argv[];
{int i;
 /* get name of tutorial file from stdin */
 if (*argc>1 && *argv[1] != '-') {
 yyin = fopen(argv[1],"r");
 if (yyin == NULL) {
 fprintf(stderr,"%s: Can't open %s\n",argv[0],argv[1]);
 exit(1);
 }
 for (i=1; i < ( *argc - 1 ); i++)
 argv[i] = argv[i+1];
 (*argc)--;
 }
 /* if name of tutorial file not given, assume */
 else {
 yyin = fopen("post.script","r");
 if (yyin == NULL) {
 fprintf(stderr,"%s: Can't open post.script\n",argv[0]);
 exit(1);
 }
 }
}
```

```
/*********************************************************************/
/* FUNCTION: create_base_frame() */
/* PURPOSE: create the main window that the entire tutorial */
/* operates */
/* CALLED BY: main() */
/* RETURNS: void */
/* CALLS: */
/* GLOBALS: Frame frame <interp.h> */
/*********************************************************************/

void create_base_frame(argc, argv)
int argc;
char *argv[];
{
 /* create icon image */
 post_icon = icon_create(ICON_IMAGE, &post_sym, 0);

 frame = (Frame)window_create(NULL, FRAME,
 FRAME_LABEL, ((topmenu != NULL) ? topmenu->title
 : "Physics of Sound Tutorial"),
 FRAME_ICON, post_icon,
 FRAME_ARGS, argc, argv,
 FRAME_NO_CONFIRM, FALSE,
 NULL);
}
```

```
/*********************************************************************/
/* FUNCTION: create_base_panel() */
/* PURPOSE: create the buttons for the main window of the */
/* tutorial */
/* CALLED BY: main() */
/* RETURNS: void */
/* CALLS: continue_selected */
/* exit_selected */
/* help_selected */
/* start_selected */
/* GLOBALS: Frame frame <interp.h> */
/* Panel menu_panel, panel <interp.h> */
/*********************************************************************/

void create_base_panel()
{
 panel = (Panel)window_create(frame, PANEL,
 WIN_WIDTH, WIN_EXTEND_TO_EDGE,
 NULL);

 menu = (Panel_item)panel_create_item(panel, PANEL_BUTTON,
 PANEL_LABEL_IMAGE, panel_button_image(panel, "START", 5, 0),
 PANEL_NOTIFY_PROC, start_selected,
 NULL);

 help = (Panel_item)panel_create_item(panel, PANEL_BUTTON,
 PANEL_LABEL_IMAGE, panel_button_image(panel, "HELP", 4, 0),
 PANEL_NOTIFY_PROC, help_selected,
 NULL);

 cont = (Panel_item)panel_create_item(panel, PANEL_BUTTON,
 PANEL_LABEL_IMAGE, panel_button_image(panel, "CONTINUE",
 8, 0),
 PANEL_NOTIFY_PROC, continue_selected,
 NULL);

 quit = (Panel_item)panel_create_item(panel, PANEL_BUTTON,
 PANEL_ITEM_X, 590,
 PANEL_LABEL_IMAGE, panel_button_image(panel, "EXIT", 4, 0),
 PANEL_NOTIFY_PROC, exit_selected,
 NULL);

 window_fit_height(panel);
```

}

```
/*********************************************************************/
/* FUNCTION: create_base_canvas() */
/* PURPOSE: create the base canvas for the main window of the */
/* tutorial */
/* CALLED BY: main() */
/* RETURNS: void */
/* CALLS: canvas_repaint */
/* GLOBALS: Frame frame <interp.h> */
/* Canvas canvas <interp.h> */
/*********************************************************************/

void create_base_canvas()
{Pixwin *pw;
 /* create a canvas on which to display the tutorial */
 canvas = (Canvas)window_create(frame, CANVAS,
 WIN_X, 10,
 WIN_COLUMNS, 60,
 WIN_ROWS, 45,
 CANVAS_AUTO_EXPAND, TRUE,
 CANVAS_AUTO_SHRINK, TRUE,
/* CANVAS_REPAINT_PROC, canvas_repaint, */
 NULL);
/*
 printf("Width of 60 chars is %d, Height of 45 lines is %d\n",
 (int) window_get(canvas,WIN_WIDTH),
 (int)window_get(canvas, WIN_HEIGHT));
*/
 pw = canvas_pixwin(canvas);
 window_set(canvas,
 WIN_CONSUME_KBD_EVENTS, WIN_NO_EVENTS, WIN_ASCII_EVENTS,
0,
 WIN_CONSUME_PICK_EVENTS, WIN_MOUSE_BUTTONS,
 WIN_IN_TRANSIT_EVENTS, 0,
 NULL);
 win_register(canvas, pw, input_events, kill_window, PW_RETAIN);
 notify_interpose_event_func(canvas, my_notify, NOTIFY_SAFE);
}
```

```
/*********************************************************************/
/* FUNCTION: start_selected() */
/* PURPOSE: provide on-line help to user */
/* CALLED BY: create_base_frame() */
/* RETURNS: int */
/* CALLS: */
/* GLOBALS: */
/*********************************************************************/

int start_selected(item, event)
Panel_item item;
Event *event;
{
 char *start_state;

 Alert_attribute *attr_list = (Alert_attribute *)malloc
 (10*sizeof(Alert_attribute));

 /* load attr_list from topmenu */
 panel_value = alert_prompt(panel, NULL,
 ALERT_MESSAGE_STRINGS, "Select starting point",
 NULL,
 ALERT_BUTTON, "Introduction", 0,
 ALERT_BUTTON, "Source", 1,
 ALERT_BUTTON, "Sound", 2,
 ALERT_BUTTON, "Detector", 3,
 ALERT_BUTTON, "Mastery Test", 4,
 NULL);

 /* 0, 1, 2, 3 and 4 are value returned when button is pushed */

 /* get main menu item selected and start point of tutuorial */
 start_state = menu_selection();

 /* find start state in cfdgraph */
 current_state = findnode(start_state);

 /* do actions in start state action list */
 do_action_list(current_state->actlist);

 /* get event and find in response list
 do_response_list(current_state->reslist);
 */
```

```
/* return to main */
}
```

```
/*********************************************************************/
/* FUNCTION: help_selected() */
/* PURPOSE: provide on-line help to user */
/* CALLED BY: create_base_frame() */
/* RETURNS: int */
/* CALLS: */
/* GLOBALS: */
/*********************************************************************/

int help_selected(item, event)
Panel_item item;
Event *event;
{
 int result;

 result = alert_prompt(panel, NULL,
 ALERT_MESSAGE_STRINGS, "Help button selected", NULL,
 ALERT_BUTTON_YES, "OK",
 ALERT_BUTTON_NO, "CANCEL",
 NULL);

 if (result == ALERT_YES)
 return 1;
 else return 0;
}
```

```
/*********************************************************************/
/* FUNCTION: continue_selected() */
/* PURPOSE: provide on-line help to user */
/* CALLED BY: create_base_frame() */
/* RETURNS: int */
/* CALLS: */
/* GLOBALS: */
/*********************************************************************/

int continue_selected(item, event)
Panel_item item;
Event *event;
{
 int result;

 result = alert_prompt(panel, NULL,
 ALERT_MESSAGE_STRINGS, "Do you wish to continue?", NULL,
 ALERT_BUTTON_YES, "YES",
 ALERT_BUTTON_NO, "NO",
 NULL);

 if (result == ALERT_YES)
 return 1;
 else return 0;
}
```

```
/*********************************************************************/
/* FUNCTION: exit_selected() */
/* PURPOSE: provide on-line help to user */
/* CALLED BY: create_base_frame() */
/* RETURNS: int */
/* CALLS: */
/* GLOBALS: */
/*********************************************************************/
void kill_window(frame, event, args)
Window frame;
Event *event;
char *args;
{
 /* this can't call window_destroy (for fear of recursive loop) */
 /* but eventually needs to do almost everything else to clean up */
 exit(0);
}

void exit_selected(argc, argv)
int *argc;
char *argv[];
{
 window_destroy(frame);
 kill_window(frame, NULL, NULL);
}
```

```
/*************************************************************************/
/* FUNCTION: menu_seclection() */
/* PURPOSE: call procedure based on main menu option selected */
/* CALLED BY: create_base_panel() */
/* RETURNS: void */
/* CALLS: */
/* GLOBALS: Panel menu_panel <interp.h> */
/*************************************************************************/

char *menu_selection()
{
 /* based on the main menu option chosen, find the start state */
 /* of the tutorial */
 switch (panel_value){
 case 0:{
 printf("INTRODUCTION chosen\n");
 printf("Go to start state: %s\n", cfdgraph->nodeid);
 return (cfdgraph->nodeid);
 break;
 }
 case 1:{
 printf("SOURCE chosen\n");
 return ("st_2");
 break;
 }
 case 2:{
 printf("MEDIUM chosen\n");
 break;
 }
 case 3:{
 printf("DETECTOR chosen\n");
 break;
 }
 case 4:{
 printf("MT chosen\n");
 break;
 }
 } /* End switch */
}
```

```
#include "interp.h"
#include <suntool/icon_load.h>

static int is_pausing = 0;
static int mousex = 0;
static int mousey = 0;
static char inbuf[2048];
static int inbuflen = 0;
static int inx = 0;
static int iny = 0;

/*****************************************************************/
/* FUNCTION: canvas_repaint() */
/* PURPOSE: repaint canvas window */
/* CALLED BY: main() <interp.c> */
/* RETURNS: void */
/* CALLS: */
/* GLOBALS: Canvas canvas <interp.h> */
/*****************************************************************/

void canvas_repaint(cvs, pwin, xrects)
Canvas cvs;
Pixwin *pwin;
int xrects; /* unused */
{
 Pixwin *pw = canvas_pixwin(canvas);

 /* EVENTUALLY - BUILD A REPAINT LIST IN DO ACTION LIST */
 /* EVENTUALLY - decode event */
 /* EVENTUALLY - interpret which state to go to based on event */
 /* EVENTUALLY - do actions in that next state */
 /* EVENTUALLY - return to main */
}
```

```
/*********************************************************************/
/* FUNCTION: do_write() */
/* PURPOSE: display text in canvas window */
/* CALLED BY: do_action_list() */
/* RETURNS: void */
/* CALLS: */
/* GLOBALS: Canvas canvas <interp.h> */
/*********************************************************************/

void do_write(dx, dy, dw, dh, str)
int dx, /* x coordinate of pixwin origin */
 dy, /* y coordinate of pixwin origin */
 dw, /* width of pixwin */
 dh; /* height of pixwin */
char *str; /* string to be printed in pixwin */
{
 char *cur;
 int pos;
 char save;
 Pixwin *pw = canvas_pixwin(canvas);

 printf("Write\n");

 /* clear the pixwin prior to writing */
 pw_writebackground(pw, dx - CHAR_TO_RASTER_X(1),
 dy - LINE_TO_RASTER_Y(1),
 dw + CHAR_TO_RASTER_X(1), dh, PIX_SRC);

 /* write string in pixwin */
 cur = str;
 /* EVENTUALLY -- Deal with scrolling windows */
 while (*cur != '\0') {
 if (strlen(cur)> RASTER_X_TO_CHAR(dw)) {
 /* output up to word break prior to dw and adjust cur accordingly*/
 pos = RASTER_X_TO_CHAR(dw);
 while (pos>0 && cur[pos]!=' ')
 pos--;
 if (cur[pos]!=' ')
 pos = RASTER_X_TO_CHAR(dw); /* no blank to break at */
 save = cur[pos];
 cur[pos] = '\0'; /* cheat */
 pw_text(pw, dx, dy, PIX_SRC, NULL, cur);
 cur[pos] = save; /* uncheat */
```

```
cur = cur + pos;
while (*cur == ' ')
cur++; /* skip whitespace after linebreak */
dy += LINE_TO_RASTER_Y(1);
}
else { /* print cur */
pw_text(pw, dx, dy, PIX_SRC, NULL, cur);
break;
}
} /* End while */
}
```

```
/*********************************************************************/
/* FUNCTION: my_notify() */
/* PURPOSE: track mouse and handle other needed events */
/* CALLED BY: */
/* RETURNS: integer */
/* CALLS: */
/* GLOBALS: Canvas canvas <interp.h> */
/*********************************************************************/
Notify_value my_notify(frame, event, arg, type)
Frame frame;
Event *event;
Notify_arg arg;
Notify_event_type type;
{int id = event_id(event);
 Notify_value value;
 if (id >= ASCII_FIRST && id <= ASCII_LAST)
 { if (id >= ' ' && id <= '~') {
 inbuf[inbuflen++] = (char) id;
 inbuf[inbuflen]='_'; /* simulate cursor */
 }
 else if (id == '\b' || id == ASCII_LAST /* DEL */) {
 inbuf[inbuflen] = '\0';
 if (inbuflen>0) inbuf[--inbuflen] = '_';
 }
 else if (id == '\024' /* CTRL-U */) {
 while (inbuflen>0) inbuf[inbuflen--] = '\0';
 inbuf[inbuflen] = '_';
 }
 else if (id == '\r' ) inbuf[inbuflen] = '\0';
 }
 else
 switch (id) {
 case LOC_STILL: /* fall through */
 case LOC_WINENTER: /* fall through */
 case LOC_WINEXIT: /* fall through */
 case LOC_DRAG: /* fall through */
 case LOC_TRAJECTORY: /* fall through */
 case LOC_RGNENTER: /* fall through */
 case LOC_RGNEXIT: /* fall through */
 case MS_LEFT: /* fall through */
 case MS_RIGHT: /* fall through */
 case MS_MIDDLE: /* fall through */
 case LOC_MOVE:
```

```
mousex = event_x(event);
mousey = event_y(event);
break;
default: break;
}
value = notify_next_event_func(frame,event,arg,type); /* let window handle */
return value;
}
```

```
/*****************************************************************/
/* FUNCTION: do_expr() */
/* PURPOSE: evaluate expression structure and return value */
/* CALLED BY: do_location() */
/* RETURNS: integer */
/* CALLS: (recursive) */
/* GLOBALS: Canvas canvas <interp.h> int mousex, mousey */
/*****************************************************************/
int do_expr(exptr)
struct expnode *exptr;
{int lhs, rhs;
 if (exptr == NULL) return 0;
 if (exptr->left == NULL && exptr->right == NULL) {
 if (exptr->varname != NULL) {
 if (strcmp(exptr->varname, "halfwid") == 0)
 return (int) window_get(canvas, WIN_WIDTH)/6;
 else if (strcmp(exptr->varname, "halfht") == 0)
 return (int) window_get(canvas, WIN_HEIGHT)/6;
 else if (strcmp(exptr->varname, "mouseX") == 0)
return mousex;
 else if (strcmp(exptr->varname, "mouseY") == 0)
return mousey;
 /* EVENTUALLY: search for varname in redisplay list & deal with
 components*/
 }
 else return exptr->val;
 }
 lhs = 0;
 rhs = 1;
 if (exptr->left != NULL) lhs = do_expr(exptr->left);
 if (exptr->right != NULL) rhs = do_expr(exptr->right);
 switch (exptr->op) {
 case ' ': return lhs;
 case '-': return lhs-rhs;
 case '*': return lhs*rhs;
 case '/': return lhs/rhs;
 case '+':
 default : return lhs+rhs;
 }
}
```

```
/*************************************************************************/
/* FUNCTION: do_location() */
/* PURPOSE: determine location for actions in list */
/* CALLED BY: do_action_list() */
/* RETURNS: x,y */
/* CALLS: do_expr */
/* GLOBALS: <interp.h> *
/*************************************************************************/
void do_location(locptr, x, y)
struct locnode *locptr;
int *x, *y;
{if (locptr == NULL) return;
 else {
 *x = do_expr(locptr->x);
 *y = do_expr(locptr->y);
 }
}
```

```
/*********************************************************************/
/* FUNCTION: do_action_list() */
/* PURPOSE: execute actions in action list */
/* CALLED BY: start_selected() <interp.c> */
/* RETURNS: void */
/* CALLS: do_location() */
/* GLOBALS: Canvas canvas <interp.h> */
/*********************************************************************/
void do_action_list(action_node)
struct actnode *action_node;
{
 int dx, /* x coordinate of pixwin origin */
 dy, /* y coordinate of pixwin origin */
 dw, /* width of pixwin */
 dh, /* height of pixwin */
 op; /* rasterop */
 Pixwin *pw = canvas_pixwin(canvas);
 Pixrect *image;
 char error_msg[IL_ERRORMSG_SIZE];

 for (inbuflen = 0; inbuflen <2048; inbuflen++)
 inbuf[inbuflen] = '\0';
 inbuflen = 0;

#define REG_WIDTH (int) window_get(canvas, WIN_WIDTH)/3
#define REG_HEIGHT (int) window_get(canvas, WIN_HEIGHT)/3
 while (action_node != NULL){
 switch (action_node->actloc){
 case REG_ALL:{ /* region 0 */
 printf("Action in region ALL (code = %d)\n",
 action_node->actloc);
 dx = 0;
 dy = LINE_TO_RASTER_Y(1);
 dw = (int) window_get(canvas, WIN_WIDTH);
 dh = (int) window_get(canvas, WIN_HEIGHT);
 break;
 }
 case REG_ONE:{ /* region 1 */
 printf("Action in region 1 (code = %d)\n", action_node->actloc);
 dx = 0;
 dy = LINE_TO_RASTER_Y(1);
 dw = REG_WIDTH;
 dh = REG_HEIGHT;
```

```
 break;
 }
 case REG_TWO:{
 printf("Action in region 2 (code = %d)\n", action_node->actloc);
dx = REG_WIDTH;
dy = LINE_TO_RASTER_Y(1);
dw = REG_WIDTH;
dh = REG_HEIGHT;
 break;
 }
 case REG_THREE:{
 printf("Action in region 3 (code = %d)\n", action_node->actloc);
dx = 2 * REG_WIDTH;
dy = LINE_TO_RASTER_Y(1);
dw = REG_WIDTH;
dh = REG_HEIGHT;
 break;
 }
 case REG_FOUR:{
 printf("Action in region 4 (code = %d)\n", action_node->actloc);
dx = 0;
dy = REG_HEIGHT + LINE_TO_RASTER_Y(1);
dw = REG_WIDTH;
dh = REG_HEIGHT;
 break;
 }
 case REG_FIVE:{
 printf("Action in region 5 (code = %d)\n", action_node->actloc);
dx = REG_WIDTH;
dy = REG_HEIGHT + LINE_TO_RASTER_Y(1);
dw = REG_WIDTH;
dh = REG_HEIGHT;
 break;
 }
 case REG_SIX:{
 printf("Action in region 6 (code = %d)\n", action_node->actloc);
dx = 2 * REG_WIDTH;
dy = REG_HEIGHT + LINE_TO_RASTER_Y(1);
dw = REG_WIDTH;
dh = REG_HEIGHT;
 break;
 }
 case REG_SEVEN:{
```

```
 printf("Action in region 7 (code=%d)\n", action_node->actloc);
dx = 0;
dy = 2 * REG_HEIGHT + LINE_TO_RASTER_Y(1);
dw = REG_WIDTH;
dh = REG_HEIGHT;
 break;
 }
 case REG_EIGHT:{
 printf("Action in region 8 (code=%d)\n", action_node->actloc);
dx = REG_WIDTH;
dy = 2 * REG_HEIGHT + LINE_TO_RASTER_Y(1);
dw = REG_WIDTH;
dh = REG_HEIGHT;
 break;
 }
 case REG_NINE:{
 printf("Action in region 9 (code=%d)\n", action_node->actloc);
dx = 2 * REG_WIDTH;
dy = 2 * REG_HEIGHT + LINE_TO_RASTER_Y(1);
dw = REG_WIDTH;
dh = REG_HEIGHT;
 break;
 }
 case REG_OTHER:
 default: {
 printf("Unrecognized region code %d\n",action_node->actloc);
 }
 } /* End switch */

 switch (action_node->action){
 case ACT_DRAW:{
 printf("Draw\n");
 if (image = icon_load_mpr(action_node->info_str,
 error_msg)) {
 do_location(action_node->info_loc, &dx, &dy);
 pw_write(pw, dx, dy, 64, 64, PIX_SRC, image, 0, 0);
 }
 else fprintf(stderr, "%s\n", error_msg);
 break;
 }
 case ACT_CLEAR:{
 printf("Clear\n");
 if (action_node->info_str != NULL) {
```

```
do_location(action_node->info_loc, &dx, &dy);
pw_writebackground(pw, dx, dy, 64, 64, PIX_SRC);
}
else pw_writebackground(pw, dx - CHAR_TO_RASTER_X(1),
dy - LINE_TO_RASTER_Y(1),
dw + CHAR_TO_RASTER_X(1), dh, PIX_SRC);
break;
}
case ACT_WRITE:{
do_write(dx,dy,dw,dh,action_node->info_str);
break;
}
case ACT_INPUT:{
printf("Input\n");
inx = dx;
iny = dy;
pw_writebackground(pw, dx - CHAR_TO_RASTER_X(1),
dy - LINE_TO_RASTER_Y(1),
dw + CHAR_TO_RASTER_X(1), dh, PIX_SRC);
break;
}
case ACT_PAUSE:{
printf("Pause\n");
sleep(action_node->info_int);
/* while (is_pausing); set by pause, cleared by end_of_pause */
break;
}
case ACT_QUIT:{
printf("Quit\n");
break;
}
case ACT_DRAG:{
printf("Drag\n");
break;
}
} /* End switch */

action_node = action_node->next;
} /* End while */
/* test for only no-user-action responses & set up for seconds */

}
```

```
/*********************************************************************/
/* FUNCTION: do_response_list() */
/* PURPOSE: execute actions in response list */
/* CALLED BY: input_events() */
/* RETURNS: int */
/* CALLS: create_assert_list <do_assert_list.c> */
/* GLOBALS: current_state <interp.c>, inbuf, inbuflen */
/*********************************************************************/

int do_response_list(response_node, res_code)
struct resnode *response_node;
int res_code;
{ struct resnode *cur = response_node;
 int retval = 0;
 printf("testing %d\n",res_code);
 while (cur != NULL){
 if (res_code == RES_NULL) {
 /* evaluate expression */
 }
 else if (res_code == cur->expr->res_act) {
 /* evaluate and test event */
 }
 if (retval) {
 if (cur->label != NULL)
 create_assert_list(current_state->nodeid,
 cur->label);
 current_state = cur->node;
 cur = NULL;
 }
 else cur = cur->next;
 } /* End while */
 return retval;
}
```

```
/*********************************************************************/
/* FUNCTION: input_events() */
/* PURPOSE: handle events and call for response evaluation */
/* CALLED BY: define_base_canvas() <interp.c> */
/* RETURNS: void */
/* CALLS: do_response_list do_action_list */
/* GLOBALS: current_state <interp.c> */
/*********************************************************************/

void input_events(window, event, arg)
Window window;
Event *event;
caddr_t arg;
{int id = event_id(event);
 int rescode = RES_NULL;

#define TEST_RES(Opcode) \
 if (do_response_list(current_state->reslist, Opcode)) \
 do_action_list(current_state->actlist)

 if (current_state == NULL) return;

 if (id >= ASCII_FIRST && id <= ASCII_LAST) {
 if ((char) id == '\r') {
 TEST_RES(RES_KEY);
 for (inbuflen=0; inbuflen<2048; inbuflen++)
 inbuf[inbuflen] = '\0';
 inbuflen=0;
 }
 else do_write(inx, iny, REG_WIDTH, REG_HEIGHT, inbuf);
 }
 else if (id == MS_LEFT) {
 TEST_RES(RES_CLICKLEFT);
 else TEST_RES(RES_CLICKANY);
 }
 else if (id == MS_MIDDLE) {
 TEST_RES(RES_CLICKMID);
 else TEST_RES(RES_CLICKANY);
 }
 else if (id == MS_RIGHT) {
 TEST_RES(RES_CLICKRIGHT);
 else TEST_RES(RES_CLICKANY);
 }
```

```
else if (id == LOC_MOVE || id == LOC_DRAG || id == LOC_TRAJECTORY) {
TEST_RES(RES_MOUSEMOVE);
}
}
```

```
/*********************************************************************/
/* FUNCTION: pause() */
/* PURPOSE: start and monitor timer */
/* CALLED BY: do_action_list() */
/* RETURNS: void */
/* CALLS: */
/* GLOBALS: */
/*********************************************************************/

Notify_value pause_time(client, which)
Notify_client client;
int which;
{
printf("Entered PAUSE_TIME\n");
printf("%i seconds\n", which);
 is_pausing = 0;
 return (NOTIFY_DONE);
}

void pause(time)
int time;
{
 struct itimerval run_timer;

printf("Entered PAUSE procedure\n");
printf("Length of timer is %i\n", time);
if (time <= 0) return; /* smart-alecs... */
is_pausing = 1;
 /* set up interval with which to RELOAD the timer */
 run_timer.it_interval.tv_usec = 0;
 run_timer.it_interval.tv_sec = 0; /* timer interval */

 /* set up INITIAL value with which to set the timer */
 run_timer.it_value.tv_usec = 0;
 run_timer.it_value.tv_sec = time; /* current value */

printf("Calling set_itimer_func\n");
 /* turn on interval timer for client */
 (void)notify_set_itimer_func(canvas, pause_time, ITIMER_REAL,
 &run_timer, ITIMER_NULL);
}
```

```
#include "interp.h"

/*****************************************************************/
/* FUNCTION: create_assert_list() */
/* PURPOSE: create linked list of assert states and identifier */
/* CALLED BY: do_response_list() */
/* RETURNS: void */
/* CALLS: find_assert_list() */
/* GLOBALS: struct assert *assert_list <interp.h> */
/*****************************************************************/

struct assert *create_assert_list(node, identifier)
char *node, *identifier;
{

 struct assert *tmp = assert_list;
 char *temp_id;
 int found = 1;

 /* allocate space for temp_id */
 temp_id = (char *)malloc(strlen(node) + strlen(identifier)
 + 1);

 /* put asserted state in proper format for comparison with past */
 /* identifier */
 strcpy(temp_id, node);
 strcat(temp_id, "/");
 strcat(temp_id, identifier);


 if (tmp == NULL){
 /* empty list, create first link */
 assert_list = (struct assert *)malloc(sizeof(struct assert));
 assert_list->id = temp_id;
 assert_list->next = NULL;
 }
 else{
 /* see if identifier already in assert list */
 printf("Before call Find, found = %i\n", found);
 found = find_assert_list_id(assert_list, temp_id);
 printf("Just returned from FIND. FIND = %i\n", find_assert_list_id(assert_list,
temp_id));
 printf("Just returned from FIND. FOUND = %i\n", found);
```

```
if (found == 0){
/* identifier not in assert list */
/* find end of list and add new link */
while (tmp->next != NULL){
tmp = tmp->next;
} /* End while, end of assert list found */
/* create space in memory for new link */
tmp->next = (struct assert*)malloc(sizeof(struct assert));
tmp = tmp->next;
tmp->id = temp_id;
tmp->next = NULL;
} /* End if */
} /* End else */

printf("ASSERT LIST FOLLOWS\n");
tmp = assert_list;
while (tmp != NULL){
 printf("%s\n", tmp->id);
 tmp = tmp->next;
}
 return(assert_list);
}
```

```
/*************************************************************************/
/* FUNCTION: find_assert_list_id() */
/* PURPOSE: find given identifier in assert list */
/* CALLED BY: create_assert_list() */
/* RETURNS: int */
/* CALLS: */
/* GLOBALS: */
/*************************************************************************/

int find_assert_list_id(list, target)
struct assert *list;
char *target;
{

int status;
printf("Begin FIND\n");
printf("TARGET is: %s and ASSERT ID is: %s\n", target, list->id);
 if (strcmp(list->id, target) == 0){ /* found match */
printf("Assert ID FOUND. Return 1.\n");
status = 1;
printf("Returning %i\n", status);
 return (status);
 }
 else{
 if (list->next == NULL){ /* end of assert list */
 /* target not found */
printf("Assert ID NOT FOUND and END OF LIST. Return 0.\n");
status = 0;
printf("Returning %i\n", status);
 return (status);
 }
 else{ /* target nof found, move to next in assert list */
printf("Assert ID NOT FOUND. Recursing. . .\n");
 return find_assert_list_id(list->next, target);
 }
 }
}
```

APPENDIX F

DATA STRUCTURE EXPLANATION

Illustrations of the menu and cfdgraph data structures appear in Figures 1 and 2, respectively, as visual

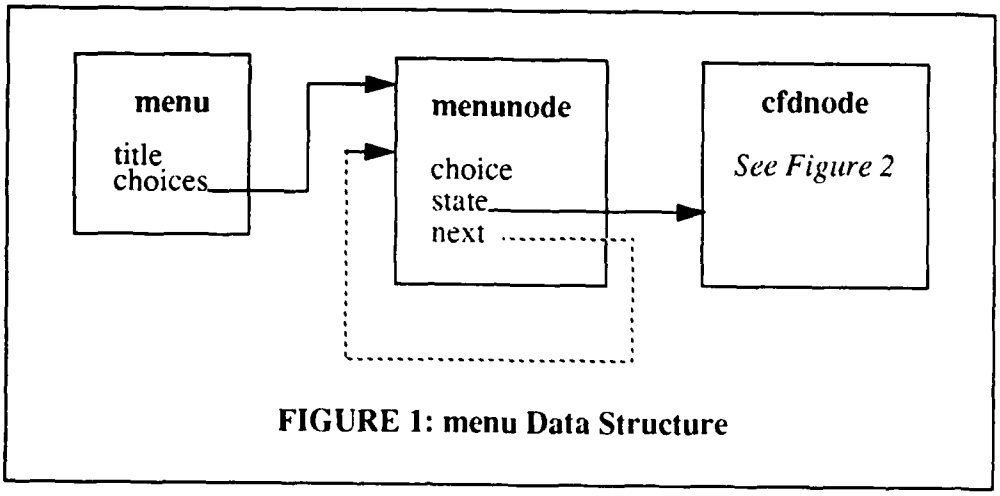

FIGURE 1: menu Data Structure

aids.

1. **menu**

   a. ***title***

      A character string of the title of the tutorial.

   b. ***choices***

      A pointer to a linked list of the titles of the start states to appear in the main menu. The **choices** pointer points to **menunode**.

2. **menunode**

   a. ***choice***

      A character string of the title of the main menu item.

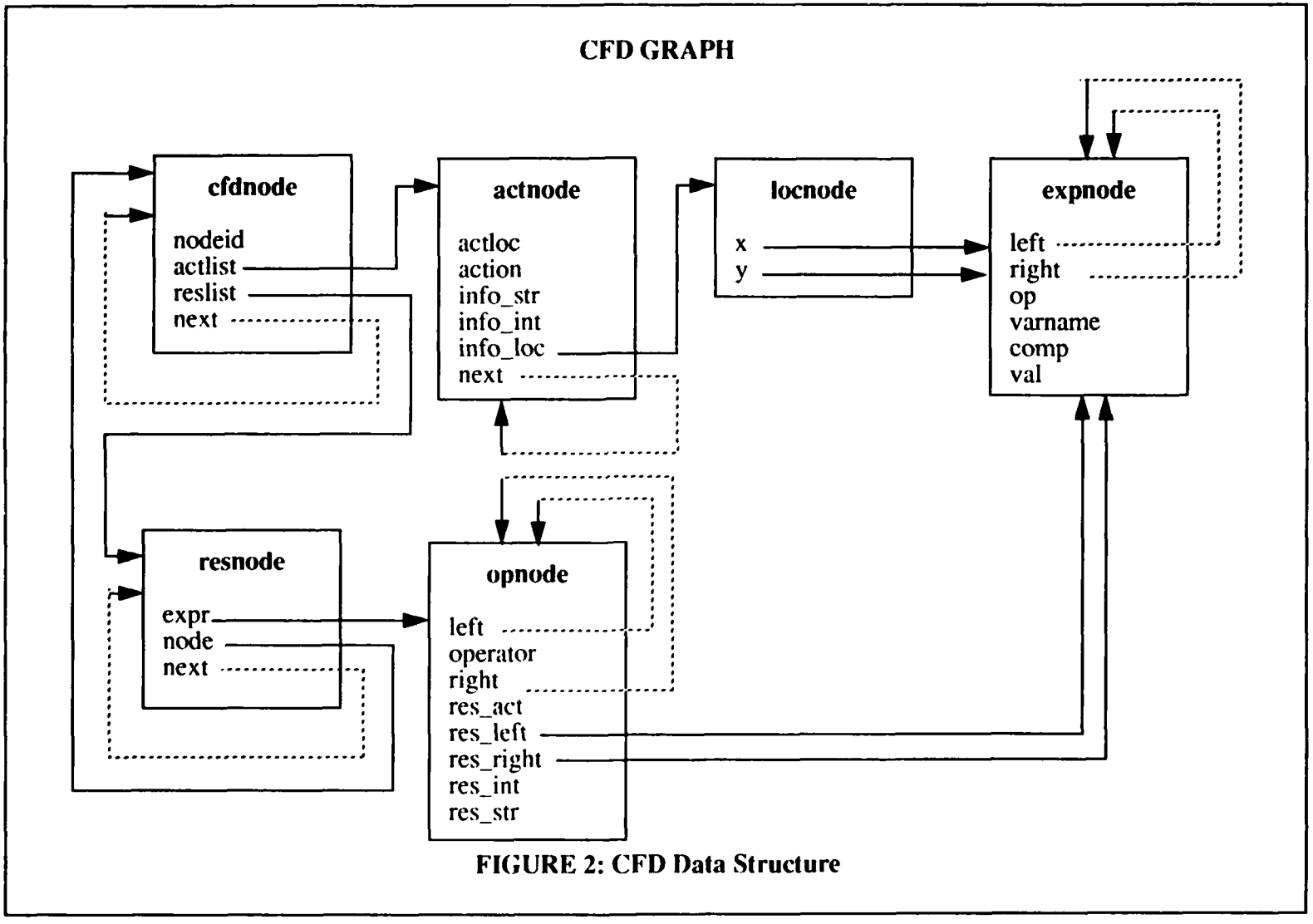

FIGURE 2: CFD Data Structure

b. state

A pointer to the state (**cfdnode**) of that main menu item in the **cfdgraph**. The **state** pointer points to **cfdnode**. This allows the tutorial to go directly to that state within the **cfdgraph**.

c. next

A pointer to the next **menunode** in the **choices** linked list.

3. **cfdnode**

a. nodeid

A character string identifying the state.

b. actlist

A pointer to a linked list of the action list. The **actlist** pointer points to an **actnode** linked list data structure.

c. reslist

A pointer to a linked list of the response list. The **reslist** pointer points to a **resnode** linked list data structure.

d. next

A pointer to a linked list of nodes in the **cfdgraph**. The **next** pointer points to the next **cfdnode** created, not necessarily the next state.

4. **actnode**

a. actloc

An integer representing the ***region_id***. The ***region_id*** and their corresponding region codes are listed in the file parser.h found in Appendix D.

b. action

An integer representing the ***action*** in the ***action_node***. The action codes are found in the file parser.h in Appendix D.

c. info_str

A character string argument for the text to be displayed in the window or the file name of an icon.

d. info_int

An integer representing the different input modes, such as mouse. The input modes and the corresponding codes are found in the file parser.h in Appendix D.

e. info_loc

A pointer to a linked list of the location arguments in the action list. The **info_loc** pointer points to a **locnode** linked list data structure.

f. next

A pointer to a linked list of each action in the action list. The **next** pointer points to **actnode**.

5. **locnode**

a. x

A pointer to a linked list data structure of the x coordinate of a location in the window. The x pointer points to an **expnode**.

b. y

A pointer to a linked list data structure of the y coordinate of a location in the window. The y pointer points to an **expnode**.

6. **expnode**

a. left

A pointer to a linked list of the operands of an arithmetic expression. The **left** pointer points to **expnode**.

b. right

A pointer to a linked list of the operands of an arithmetic expression. The **right** pointer points to **expnode**.

c. op

A character representing the four arithmetic operators: +, -, * and /.

d. varname

A character string of the argument mouseX, mouseY, the identifier for the icon file name or the variable name used in the arithmetic expression. Refer to Figures 4 and 5 for an example.

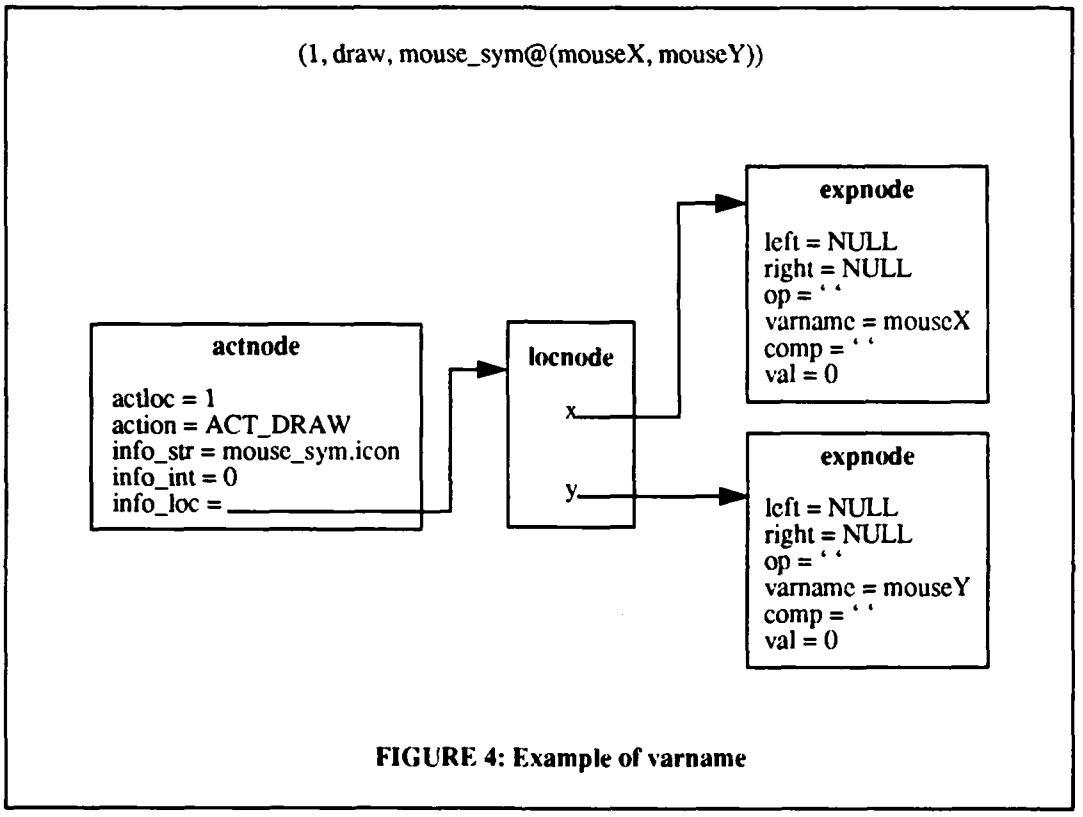

FIGURE 4: Example of varname

e. comp

A char specifying x, y, or blank. The x and y refer to ***identifier.x*** and ***identifier.y***, respectively. Refer to Figures 4 and 5 for an example.

f. val

An integer representing the operand value in the expression.

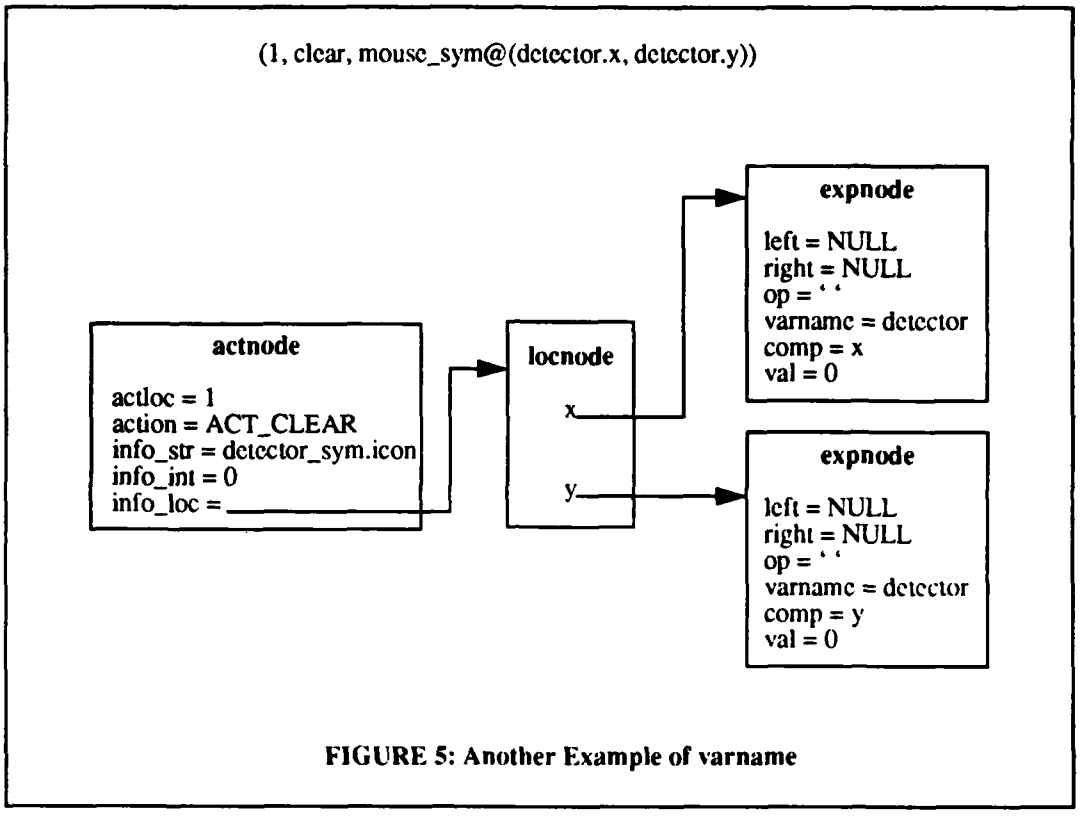

FIGURE 5: Another Example of varname

7. **resnode**

a. expr

A pointer to a tree representation of each response in the response list. The **expr** pointer points to an **opnode** linked list data structure.

b. label

A character string for the identifier associated with *assert*. When the actual assert list is created, the character '/' separates the state and the label. For example, if the state = st_1 and the label = "left," then the assert list identifier would be "st_1/left."

c. node

A pointer to the state or node in the **cfdgraph** to go to if the given response is received. The **node** pointer points to a **cfdnode** linked list data structure.

d. next

A pointer to the next response in the response list. The **next** pointer points to a **resnode** linked list data structure.

8. opnode

a. left

A pointer to the left node in a tree structure of a response that includes logical operators. The **left** pointer points to another **opnode**. Figure 20 illustrates one of the response events of state st_1_1_25 of the POST script (Appendix B).

b. right

A pointer the right node in a tree structure of a response that includes logical operators. The **right** pointer points to another **opnode**. Figure 6 illustrates one of the response events of state st_1_1_25 of the POST script (Appendix B).

c. operator

An integer representing the logical operators. The logical operators and their corresponding codes are listed in the file parser.h in Appendix D.

d. res_act

An integer representing the response event. The response codes are listed in the file parser.h found in Appendix C.

e. res_left

A pointer to an arithmetic operation used in the ***relop*** or ***loc_part*** of the grammar. The res_left pointer points to an **expnode**.

f. res_right

A pointer to an arithmetic operation used in the ***relop*** or ***loc_part*** of the grammar. The res_right pointer points to an **expnode**.

g. res_int

An integer for the integer argument in the ***response_node*** "seconds."

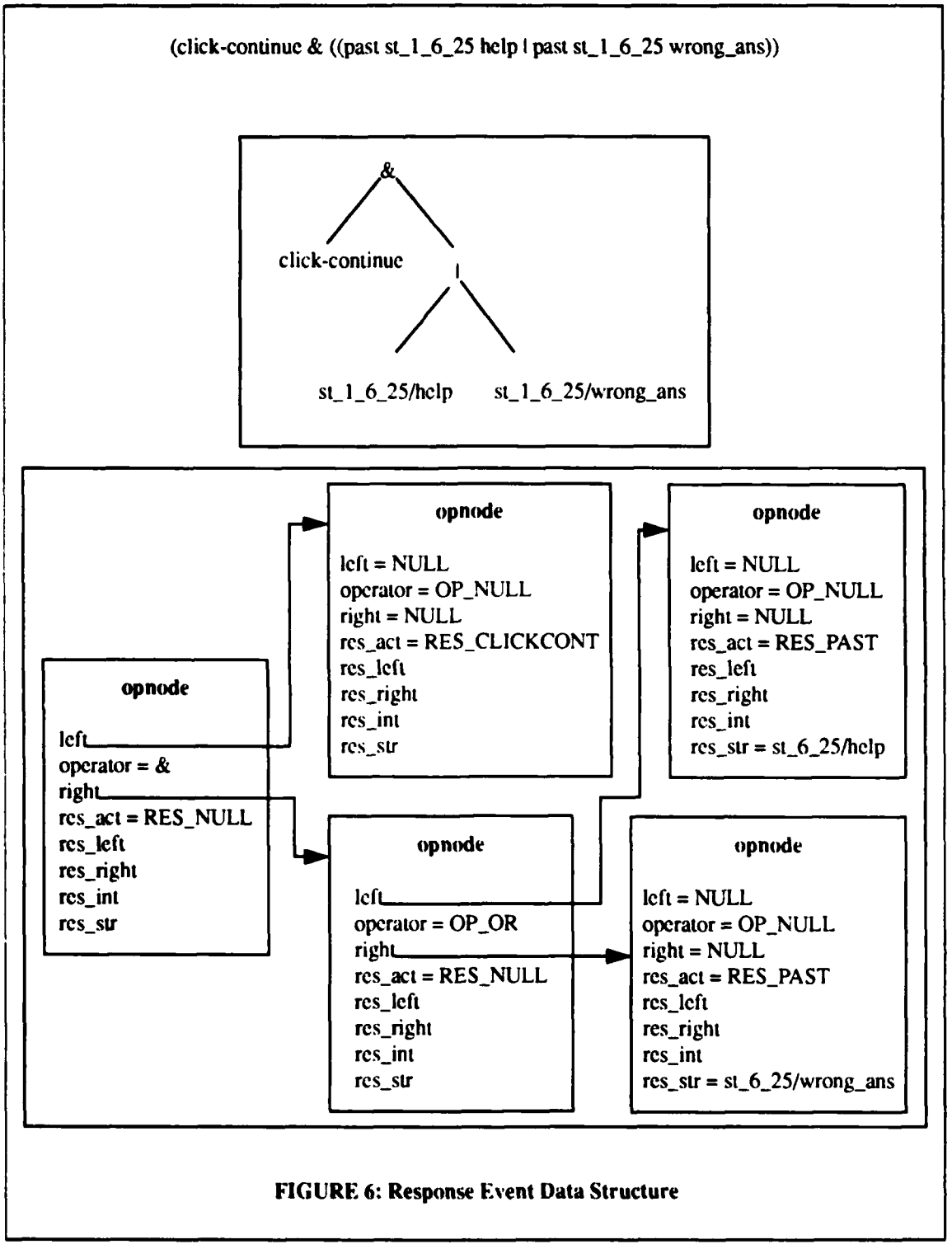

FIGURE 6: Response Event Data Structure

h. res_str

A character string representing the identifier for *past*.

LIST OF REFERENCES

Bork, Alfred, *International Development of Technology-Based Learning Courses*, Educational Technology Center, Information and Computer Science, University of California, Irvine, August 31, 1990.

Bork, Alfred, *Pedagogical Development of Computer-Based Learning Material*, Educational Technology Center, Information and Computer Science, University of California, Irvine.

Bork, Alfred, *Production Systems for Computer-Based Learning*, University of California, Irvine.

Burke, Robert L., *CAI Sourcebook*, Prentice-Hall, Inc., 1982.

Cleveland, Bernard F., *Mastering Teaching Techniques*, The Connecting Link Press, 1986.

Electronic mail from Alfred Bork, bork@idyllwild.ics.usi.edu, 08 February 1991.

Krendl, Kathy A. and Lieberman, Debra A., "Computers and Learning: A Review of Recent Research," *Journal of Educational Computing Research*, v. 4, n. 4, Baywood Publishing Co., Inc., 1988.

Levin, Tamar and Long, Ruth, *Effective Instruction*, The Association for Supervision and Curriculum Development, 1981.

Marks, Gary H. and Bartholomew, Rolland, "Computer-Assisted Instruction in Secondary Physical Science: An Evaluation of Student Achievement and Attitudes," *Proceedings of National Educational Computing Conference (NECC), 17-19 June 1981, North Texas State University*, University of Iowa, 1981.

Walker, Decker F. and Hess, Robert D., ed., *Instructional Software Principles and Perspectives for Design and Use*, Wadsworth Publishing Company, 1984.

Yourdon, E., *Modern Structured Analysis*, Prentice-Hall, Inc., 1989.

BIBLIOGRAPHY

Bork, Alfred, *International Development of Technology-Based Learning Courses*, Educational Technology Center, Information and Computer Science, University of California, Irvine, August 31, 1990.

Bork, Alfred, *Pedagogical Development of Computer-Based Learning Material*, Educational Technology Center, Information and Computer Science, University of California, Irvine.

Bork, Alfred and Pomicter, Nancy, "Practical Techniques Useful in Authoring Technology-Based Learning Material", *Journal of Computer-Based Instruction*, Spring 1990, v. 17, n. 2, 53-60, 1990.

Bork, Alfred, *Production Systems For Computer-Based Learning*, University of California, Irvine.

Bork, Alfred, *Tools For Developing Technology-Based Learning Units*, Educational Technology Center, Information and Computer Science, University of California, Irvine, May 30, 1990.

Burke, Robert L., *CAI Sourcebook*, Prentice-Hall, Inc., 1982.

Cleveland, Bernard F., *Mastering Teaching Techniques*, The Connecting Link Press, 1986.

Electronic mail from Alfred Bork, bork@idyllwild.ics.usi.edu, 08 February 1991.

Gray, Susan H., "The Effect of Locus of Control and Sequence Control on Computerized Information REtrieval and Retention," *Journal of Educational Computing Research*, v. 5, n. 4, Baywood Publishing Co., Inc., 1989.

Hativa, Nira, "Differential Characteristics and Methods Underlying CAI/CMI Drill and Practice Systems," *Journal of Research on Computing in Education*, v. 20, n. 3, Spring 1988.

Krendl, Kathy A. and Lieberman, Debra A., "Computers and Learning: A Review of Recent Research," *Journal of Educational Computing Research*, v. 4, n. 4, Baywood Publishing Co., Inc., 1988.

Levin, Tamar and Long, Ruth, *Effective Instruction*, The Association for Supervision and Curriculum Development, 1981.

Marks, Gary H. and Bartholomew, Rolland, "Computer-Assisted Instruction in Secondary Physical Science: An Evaluation of Student Achievement and Attitudes," *Proceedings of National Educational Computing Conference (NECC)*, 17-19 June 1981, North Texas State University, University of Iowa, 1981.

Mason, Tony and Brown, Doug, *lex & yacc*, O'Reilly and Associates, Inc., 1990.

Sleeman, D. and Brown, J. S., ed, *Intelligent Tutoring Systems*, Academic Press, Inc., 1982.

Urick, Robert J., *Principles of Underwater Sound*, 3d ed., McGraw-Hill Book Company, 1983.

Walker, Decker F. and Hess, Robert D., ed., *Instructional Software Principles and Perspectives For Design and Use*, Wadsworth Publishing Company, 1984.

Yourdon, E., *Modern Structured Analysis*, Prentice-Hall, Inc., 1989.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library Code 052 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	LT Dawn M. Maskell c/o 2 Lenox Court Longview Montville, NJ 07045-9001	1
4.	Timothy J. Shimeall Code CS/Sm Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5100	3
5.	Robert B. McGhee Chairman, Code CS Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5100	1
6.	CDR Thomas J. Hoskins Curricular Officer, Code 37 Naval Postgraduate School Monterey, CA 93943-5100	1
7.	Space and Naval Warfare Systems Command ATTN: CAPT Kirk E. Evans, USN (PD-80) Washington, D. C. 90363-5100	1

8. Space and Naval Warfare Systems Command — 1
LCDR J. L. Knecht, USN (PMW 183-11)
Washington, D. C. 90363-5100

9. Space and Naval Warfare Systems Command — 1
ATTN: LCDR P. A. Feldmann, USN (PMW 183-113)
Washington, D. C. 90363-5100

10. Space and Naval Warfare Systems Command — 1
ATTN: OTAC R. Bryan (PMW 181)
Washington, D. C. 90363-5100

11. Applied Research Laboratories — 1
The University of Texas at Austin
ATTN: Steve Houser
P. O. Box 8029
Austin, TX 78713-8029

12. Applied Research Laboratories — 1
The University of Texas at Austin
ATTN: Carol Sheppard
P. O. Box 8029
Austin, TX 78713-8029

13. Alfred Bork — 1
Educational Technology Center
Information and Computer Science
University of California
Irvine, CA 92717